P - 701

Kumaraguru College of Technology

Coimbatore – 641006

Department of Electrical And Electronics Engineering

Certificate

This is to certify that this project entitled

KNOCKOUT SWITCH

A SIMPLE MODULAR ARCHITECTURE FOR PACKET SWITCHING

has been submitted by

PAVITHRA R.	**9827C0093**
VINOD KUMAR V.	**9827C0712**
EBINEZAR J.	**9827C0074**
MURUGARAJ G.	**9827C0089**

In partial fulfillment of the requirements for the award of Degree of Bachelor of Engineering in the Electrcal and Electronics Engineering Branch of the Bhatrathiar University, Coimbatore - 641006 during the academic year 2001-02

(Guide)

V. DURAISAMY, M.E., M.I.S.T.E.,
Assistant Professor of Elec. Engg.
Kumaraguru College of Tech.
COIMBATORE - 641 006.
(Head of Department)

Certified that the candidates were examined by us in the Project Work.

Viva - Voce Examination held on _____

(Internal Examiner)

(External Examiner)

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

Any endeavor can be accomplished only with the benevolence of well wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We are highly indebted to our revered Principal **Dr. K. K. Padmanabhan**, B.Sc. (Engg), M. Tech, Ph.D., for his kind patronage in providing the infrastructural facilities.

Our heartfelt thanks to **Prof. V. Duraisamy** M.E., MISTE, Assistant Professor, Dept of Electrical and Electronics Engg, a highly intellectual personality for allowing us to make judicious use of various facilities in the department.

We are also immensely indebted to our class advisor **Mrs. Rani Thottungal** M.A., M.E., MISTE, Senior Lecturer, Dept of Electrical and Electronics Engg, a charismatic and dynamic personality for her concern and motivation through the years.

Thanks to our project guide **Miss. T. Karunambigai** B.E., Lecturer Dept of Electrical and Electronics Engg, for the motivation, which has enabled us to bring our project to a successful completion.

Our special and sincere thanks to **Dr. Lawrence Jenkins**, Professor, Indian Institute of Science, Bangalore, who has been a constant source of inspiration, guidance and encouragement.

Words can but express in a meager way, our thanks to all lab assistants, who have put in a lot of effort in order to give us the right kind of software and other lab facilities.

SYNOPSIS

SYNOPSIS

The use of high performance packet switching for building wide band integrated communication networks has received much attention.

Every type of switch interconnecting, N inputs and N outputs must perform two basic functions:

1) It must route the traffic arriving on its inputs to the appropriate outputs.

2) The switch must deal with output contention where traffic arriving at the same time on two or more inputs may be destined, for a common output.

In this project we have simulated the **knockout switch**. It is a new, high performance packet switching architecture, which is self-routing, has low latency, and is non-blocking. We have used simjava, a process based discrete event simulation package for the Java programming.

The simulation design of the concentrator has enabled us to determine the winner and loser based on a knockout tournament and the outputs are obtained at appropriate intervals by providing suitable delay time.

Through the buffer design we demonstrate how the packets arriving from the concentrator to the shifter are stored in the packet buffers and routed as outputs on a FIFO basis.

Thus from the simulation we conclude that the architecture of the knockout switch provides for a simple modular growth, is fault tolerant and exhibits easy maintenance procedures as compared to conventional contention switches.

CONTENTS

CONTENTS

CHAPTER 1

INTRODUCTION

Chapter 1

INTRODUCTION

1.1 SWITCHING TECHNIQUES

1.1.1 CIRCUIT SWITCHING

- When a message has to be sent to a certain destination, a route should be assigned and dedicated so that the message can be forwarded on that path only.
- Once the route is established that route will be dedicated for these two parties only.
- If there are other senders who need to send messages to the same destination, the messages will have to be discarded.

1.1.2 MESSAGE SWITCHING TECHNIQUE

- No dedicated path is needed.
- There is a store-and-forward method, which ensures that the messages will be chunked into blocks and goes to the switches randomly.
- When a message arrives, it will first be stored into the buffer of the switch, and then forwarded later.

Disadvantage:

- The size of the message chunks is variable and a delay will result during sending and receiving messages.

1.1.3 PACKET SWITCHING TECHNIQUE

- It has no dedicated path and so there will be no bandwidth wasted.
- The size of the packet is fixed.

1.1.4 CELL SWITCHING

- It is a kind of connection-oriented packet switching.
- Cell switching takes the advantages of both Circuit switching and Packet switching.
- When a message is about to be sent, a virtual circuit path is established. It behaves like circuit switching. On the other hand, the cell size is fixed and small which takes the advantages of packet switching as well.

1.2 SWITCH FUNCTIONALITY

- Broadband interconnection of networks, which carry all types of information, depends heavily on the feasibility of implementing real-time packet switching fabrics. Because of the high data rates involved, routing decision-making logic must be implemented entirely in hardware; software processing would be grossly incapable of keeping pace with the high rate of packet arrivals.
- In discussing packet switch interconnections, the arriving packets are of fixed time length. Such fixed-length packets are also referred to as cells.
- Two primary functions are supported by the N-input, N-output switch:

- Routing of each applied fixed-length cell to its correct destination port.
- Resolving the contention, which arises when several simultaneously arriving cells are headed to a common output port?

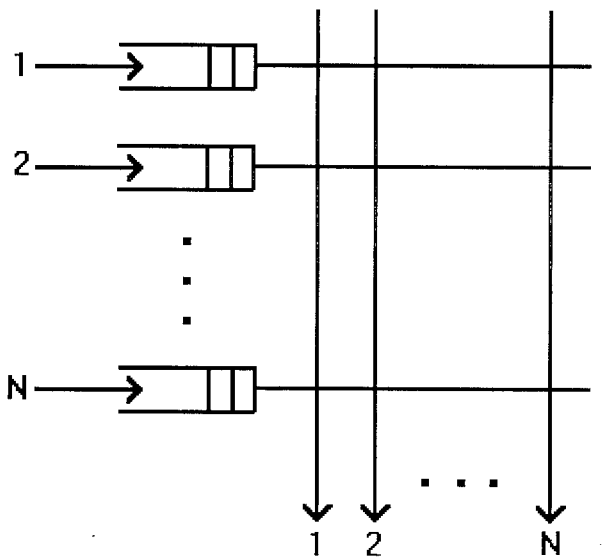
We are implicitly assuming that the destination port number is included in the cell header. Since packet switching is involved, output port contention is resolved by means of store-and-forward buffering: Arriving packets, which cannot be immediately delivered because of a contention, are stored in a buffer (or buffers) within the switch. The purpose of the buffers is to smooth out the statistical fluctuation in the arrival patterns, so that even in a case of a peak cell arrival rate followed by a quiet period, the output is a smooth continuum of cells.

1.3 THE NEED FOR QUEUING IN PACKET SWITCHES

Consider an N-input, N-output time-slotted switch. In a switch that does not contain any buffers; a cell will be lost whenever another cell simultaneously arrives on a different input and is bound for the same output. Generally, whenever k such cells arrive simultaneously, k-1 of them are lost. Without buffers for contending cells, the maximum offered load permitted for a low packet loss rate is so small that the links are grossly underutilized. Hence, smooth buffer is an important necessity.

1.4 QUEUING METHODS

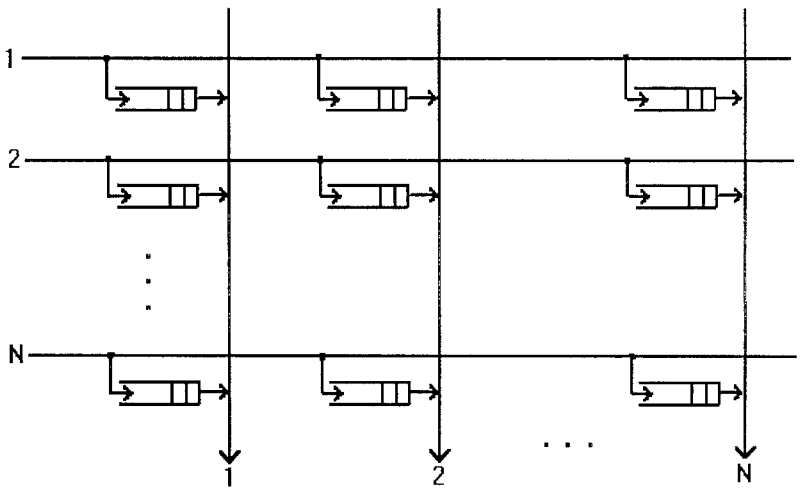
1.4.1 Input Queuing



In any given time slot, leading cells in all the buffers are sent to their output lines. If several buffer-leading cells contend for the same output, only one of them is selected according to a contention scheme (a plausible scheme may be round-robin), while the rest remain in the buffers and contend again in the very next time slot.

An input-queued N×N switch requires N^2 packet routing elements and N^2 contention elements, which can be synthesized on a single VLSI chip.

1.4.2. Output Queuing



There are N^2 buffers, one for each input-output combination. For every time slot, all cells active during that time slot enter the queue in the intersection of their input line and output line. The leading cells in all the buffers with a common output contend for that output.

The output-queued switch readily supports **broadcasting** and **multicasting**; the only difference is that an arriving cell is copied to buffers corresponding to all the outputs for which it is intended.

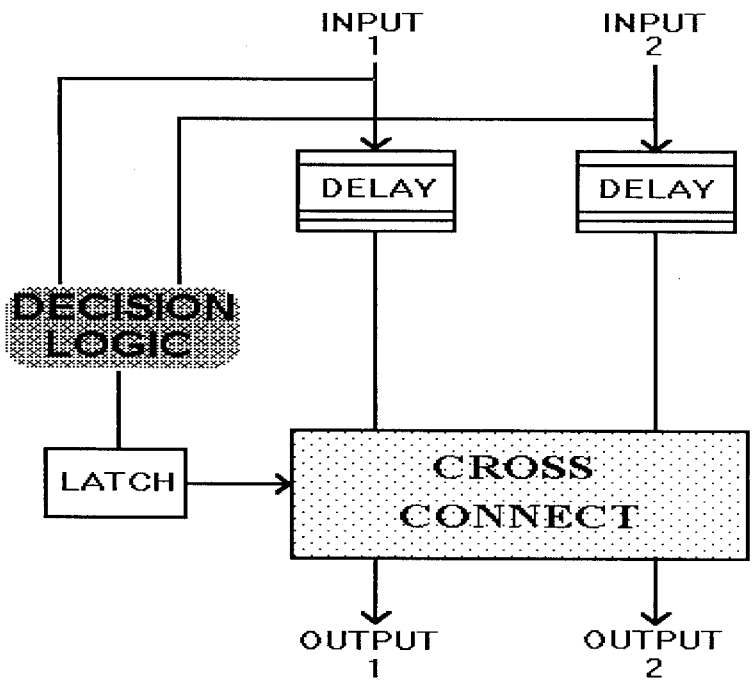
1.1. The Basic Switching Element

A 2×2 switch, is often referred to as β-element.

The structure of a β-element is depicted in the diagram.

- ◈ It consists of decision logic, which operates on the cell headers, a latch to hold the result of that decision for the duration of the cell, delay lines to synchronize the cell contents with the decision, and the 2×2 cross-connect it.

◈ The cross-connect is a dual multiplexer which can be set in either a **"bar"** or a **"cross"** state; that is, each input can be routed to either output, with the other input correspondingly routed to the other outputs.



CHAPTER 2

HARDWARE CONFIGURATION

HARDWARE CONFIGURATION

2.1 OVERVIEW OF THE KNOCKOUT SWITCH

Using the knockout switch, high performance packet switching architecture can be proposed. It has a fully interconnected switch fabric topology (i.e. each input has a direct path to every output) so that no switch blocking occurs where packets destined for one output interfere with (ie. block or delay) packets going to different outputs.

2.1.1 PRINCIPLE ADOPTED:

It is only at each output of the switch that one encounters the unavoidable congestion caused by multiple packets that simultaneously arrive on different inputs all destined for the same output.The inevitability of lost packets in a packet switching network is used to an advantage while designing the knockout switch.

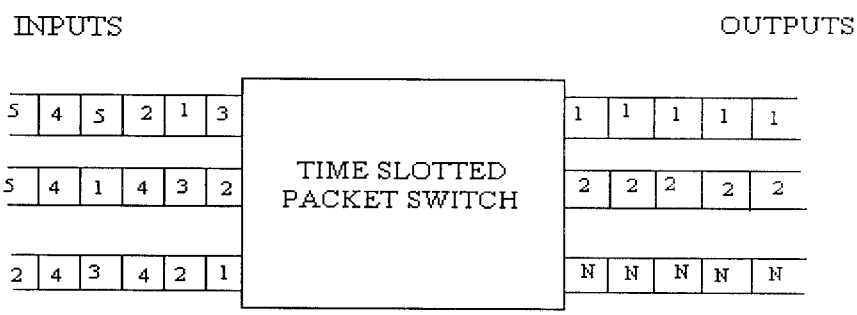
Difference between the knockout switch and the conventional contention switch

The knockout switch uses a novel concentrator design at each output to reduce the number of separate buffers needed to receive simultaneously arriving packets.

Shared buffer architecture provides complete sharing of all buffer memory at each output and ensures that all packets are placed on the output line on the FIFO basis.

2.2 KNOCKOUT SWITCH DESCRIPTION

- The knockout switch is an N-input N-output packet switch with all inputs and outputs operating at the same bit rate.
- Fixed-length packets arrive on the N-inputs in a time slotted fashion as shown in the figure, with each packet containing the address of the output port on the switch to which it is destined.
- This addressing information is used by the packet switch to route each incoming packet to its appropriate output.
- Since the output port address for an arriving packet can be determined via a look-up table prior to the packet entering the switch fabric, the Knockout switch has applications to both datagram and virtual circuit packet networks



There is no time-slot specific scheduling that prevents two or more packets from arriving on different inputs in the same time slot

destined for the same output. Hence, to avoid (or at least provide a sufficiently small probability of) lost packets, at a minimum, packet buffering must be provided in the switch to smooth fluctuations in packet arrivals destined for the same output.

2.2.1 INTERCONNECTION FABRIC:

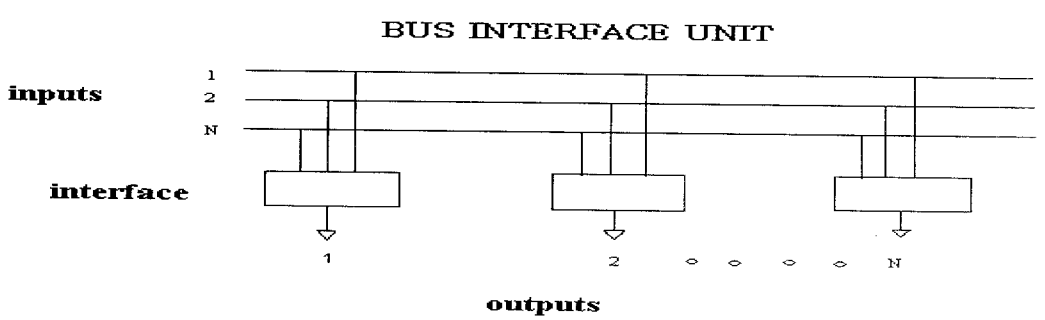
The interconnection fabric for the knockout switch has two basic characteristics:

1) Each input has a separate broadcast bus.

2) Each output has access to the packets arriving on all inputs.

The figure illustrates these two characteristics where each of the N-inputs is placed directly on a separate broadcast bus and each output passively interfaces to the complete set of N buses. This simple structure provides us with several important features:

With each input having a direct path to every output, no switch blocking occurs where packets destined for one output interfere with packets going to the other outputs.



The bus structure (Fig) has the desirable characteristic that each bus is unidirectional and is driven by only one input. This allows for a higher transmission rate by the buses, and a design more tolerant of faults compared to a shared parallel bus accessed by all inputs.

The packet buffering and bus access control circuitry of the parallel bus is replaced by, an elastic buffer at each input which is used to synchronize the time slots from the individual input lines.

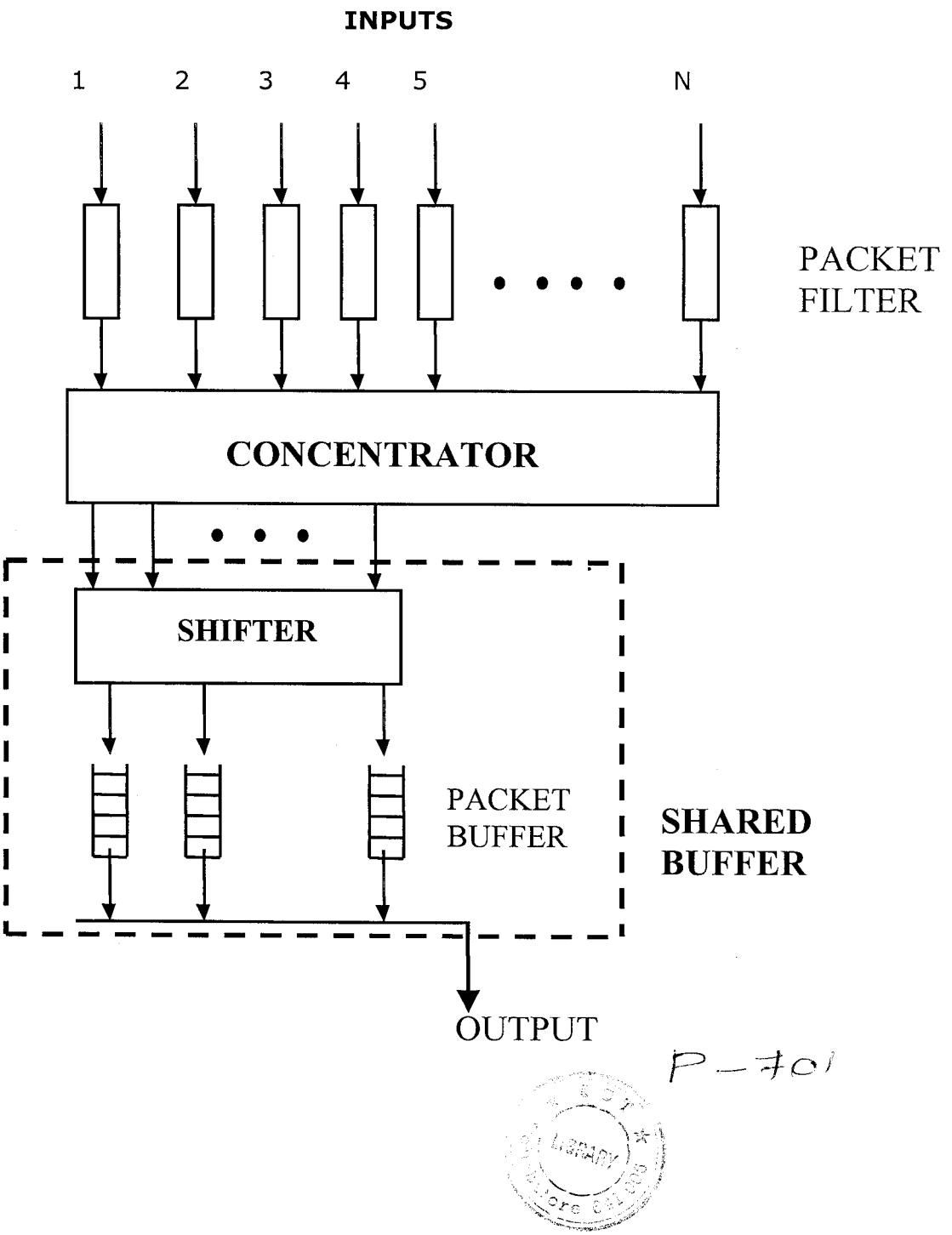
The interconnection architecture of the knockout switch lends itself to broadcast and multicast features. Since every input is available at the interface to every output, arriving packets can be addressed to and received by multiple outputs.

2.2.2 BUS INTERFACE:

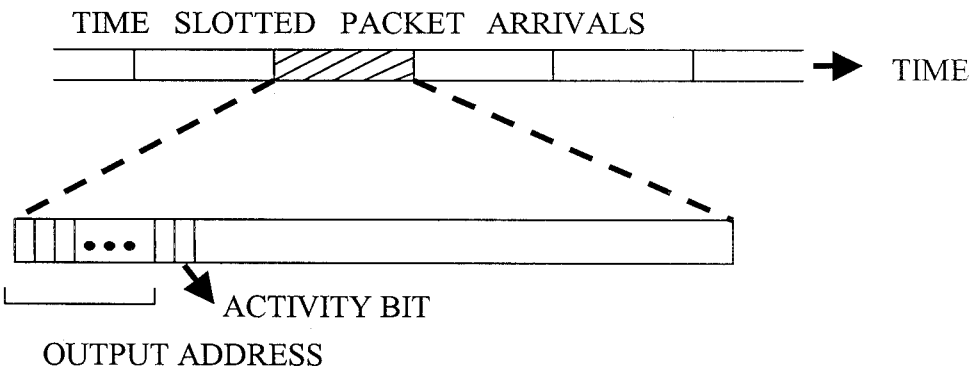
The bus interface has 3 major components:

- At the top of the fig there are a row of N packet filters. Here the address of every packet broadcast on each of the N buses is examined, with packets addressed to the outputs allowed to pass on to the concentrator and all others blocked.
- The concentrator then achieves an N to L (L<<N) concentration of the input lines, wherein up to L packets making it through the packet filters in each time slot will emerge at the outputs of the concentrator. These L concentrator outputs enter a shared buffer composed of a shifter and L separate FIFO buffers.

◈ The shared buffer allows complete sharing of the L FIFO buffer and provides the equivalent of a single queue with L inputs and one output, operating under a first-in-first-out queuing discipline.



2.2.3 PACKET FILTERS:



PACKET FORMAT

Fig shows the format of packets as they enter the packet filters from the broadcast buses.

- The beginning of each packet contains the address of the output on the switch for which the packet is destined, followed by a single activity bit.
- The destination output address contains $\log_2 N$ bits with each output having a unique address.
- The activity bit indicates the presence (logic 1) or absence (logic 0) of the packet in the arriving time slot and plays an important role in the operation of the concentrator.

At the start of every time slot the path through each of N packet filters is open, Initially allowing all arriving packets to pass through to the concentrator.

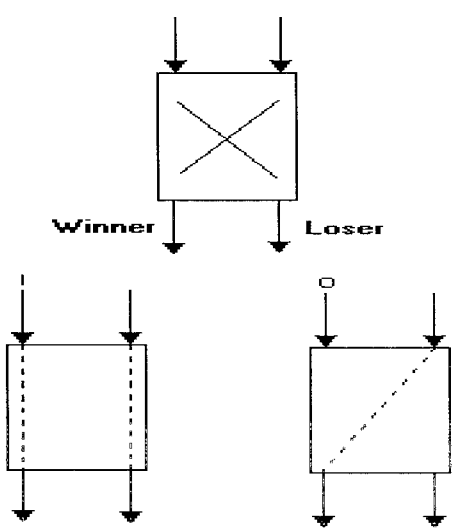
As the address bits for each arriving packet enter the row of N packet filters, they are compared bit-by-bit against the output address for the bus interface.

If at any time the address for an arriving packet differs from that of bus interface the further progress of the packet to the concentrator is blocked.

The output of the filter is set at logic 0 for the remainder of the time slot. By the end of the output address the filter will have either blocked packet, and hence also set its activity bit to 0, or, if the addresses matched, allowed the packet to continue on to the concentrator.

Even though a portion of the address bit of a blocked packet may pass through the filter and into the concentrator, these bits no longer serve any useful function and are ignored by the rest of the bus interface.

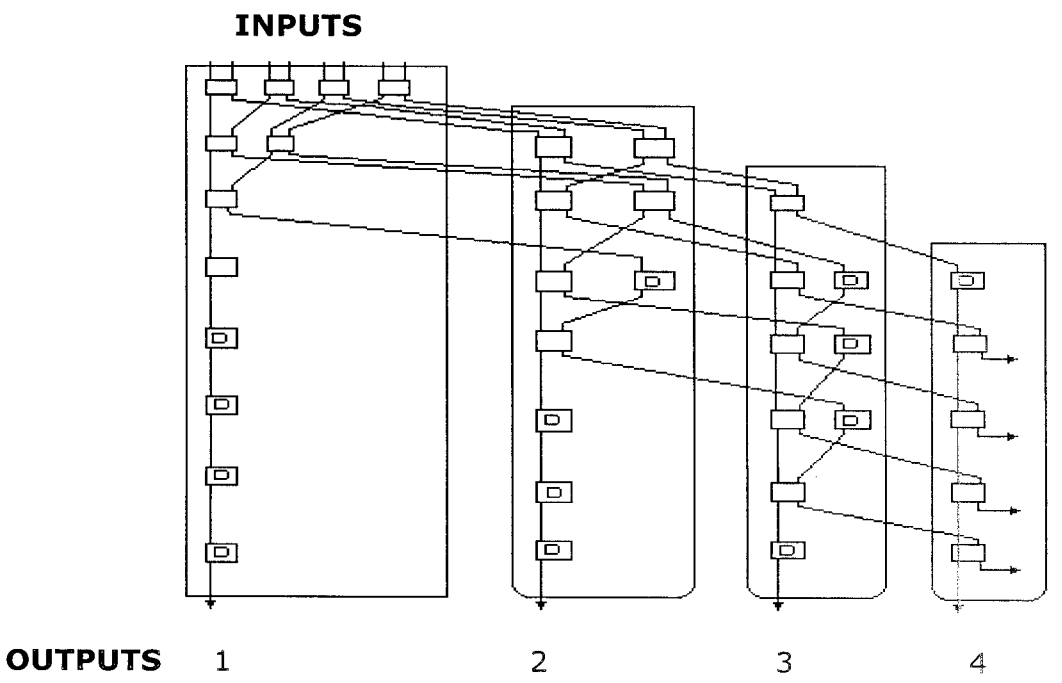
2.2.4 CONCENTRATOR:



- All packets making it through the packet filters enter the Concentrator, which achieves an N to L concentration.
- The basic building block of the concentrator is a simple 2X2

Contention switch shown in the fig. The two inputs contend for the "winner" output according to their activity bits. If only one input has an arriving packet (indicated by activity bit = 1), which is routed to the winner (left) output. If both inputs have arriving packets, one input is routed to the winner output and the other input is routed to the loser output.

- If the activity bit is 1, the left input is routed to the winner output and the right input is routed to the loser output. If the activity bit is 0, the right input is routed to the winner output and no path is provided through the switch for the left input.



- Fig shows the design of 8-input 4-output concentrator composed of these simple 2X2 switch elements and single-input /single-output 1-bit delay elements (marked by "D").
- One may view this first stage of switching as the first round of a tournament with N players, where the winner of each match emerges from the left side of the 2X2 switch element and loser emerges from the right side. The N/2 winners from the first round advance to the second round when they compete in pairs as before using a row of N/4 switch elements.
- The winners in the second round advance to the third round and this continues until two compete for the championship: that is, the right to exit the first output of the concentrator. If there is at least one packet arriving on an input to the concentrator a packet will exit the first output of the concentrator.
    1) A tournament with a single tree-structured competition leading to a single winner is sometimes referred to as a single knockout tournament: If one match is lost you are knocked out of the tournament.
    2) In a double knockout tournament the N-1 losers from the first section of the competition compete in a second section, which produces a second place finisher (second output for the concentrator) and N-2 losers.

3) As fig illustrates the loser from the first section can begin competing in the second section before the competition is finished in the first.

4) Whenever there are an odd number of players in a round, one player must wait and complete in a later round in the section. In the concentrator a simple delay element serves this function.

For a concentrator with N inputs and L outputs, there are L sections of competition ,one for each output. A packet entering the concentrator is given L opportunities to exit through a concentrator output: a packet losing L times is knocked out of the competition and is discarded by the concentrator. In all cases, however, packets are lost ,only if more than L arrive in any one time slot.

The 128-to-8 concentrator constructed from 32-to-8 concentrator chips

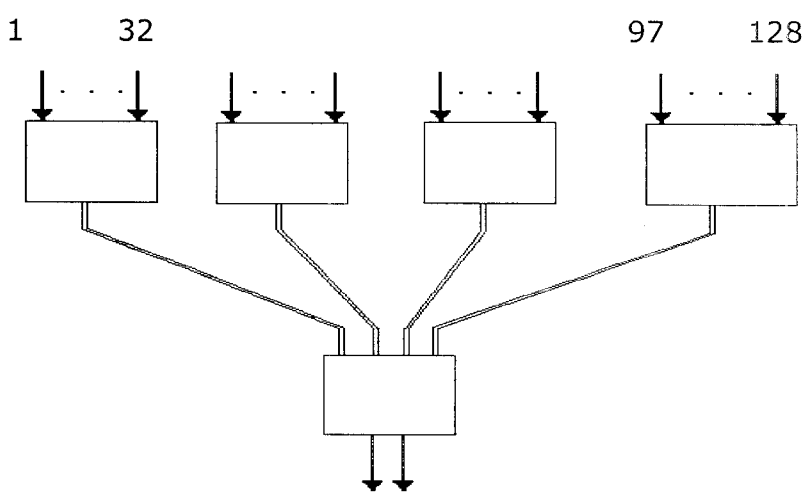


Fig illustrates how several identical chips can be interconnected to form a larger concentrator.

2.2.5 SHARED BUFFER:

1) Through the use of shared buffer structure, complete sharing of all packet buffer memory within the bus interface is made possible.

2) This is accomplished while still providing a first-in-first-out queuing discipline for the arriving packets and keeping the latency through the bus interface to a minimum.

3) Since in any given time slot up to L packets can emerge from the concentrator, the buffer within the bus interface must be capable of storing up to L packets within a single time slot.

4) To permit high-speed low-latency operation of knockout switch, the bus interface uses L separate FIFO buffers as shown in the fig.

5) A simple technique allows complete sharing of L buffers, and at the same time provides a first-in-first-out queuing discipline for all packets arriving to the output and, more importantly, that successive packets arriving on an input do not get out of sequence within the switch.

6) As shown in the bus interface block diagram, the L outputs from the concentrator first enter a "shifter" having L-inputs and L-outputs.

7) The purpose of shifter is to provide a circular shift of inputs to the outputs so that the L separate buffers are filled in the cyclic fashion.

the switch is being repaired, all other inputs and outputs can operate as usual. This is in direct contrast to a multistage switch where a failed intermediate switch point can affect multiple input/output paths and may also be difficult to locate.

2.3 FAULT TOLERANCE

In many switching applications, stringent operational requirements necessitate a design that is tolerant of faults. This may be achieved by duplicating the entire switch fabric to serve as a standby in the event of a failure. With the knockout switch, since all interface modules are identical, a fault tolerant design can be achieved by providing only a single spare interface module attached to the N broadcast buses. This spare module could take over the operation of any of the N interface modules if a failure occurs. Again, service would only be disrupted on the input or output attached to the failed module, and only as long as it takes to locate the fault and switch over its input and output to the spare module.

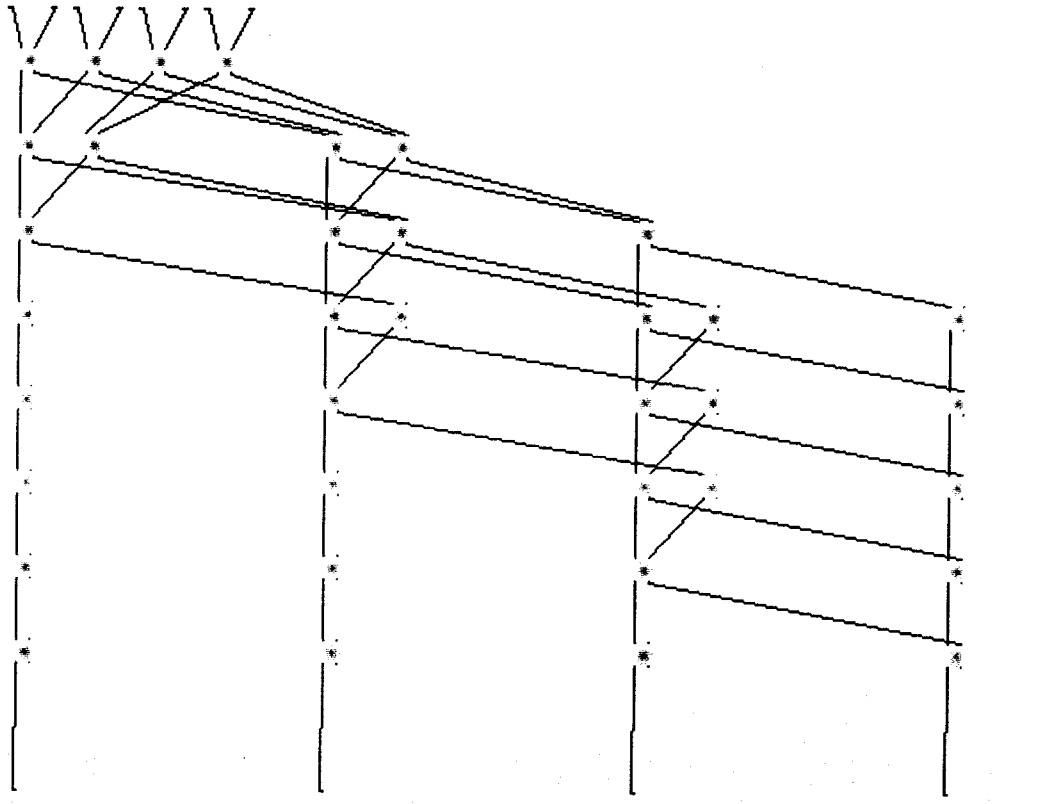
CHAPTER 3

DESIGN SIMULATION

DESIGN OF 8-INPUT BY 4-OUTPUT CONCENTRATOR

Applet Viewer: Concentrator.class

Applet



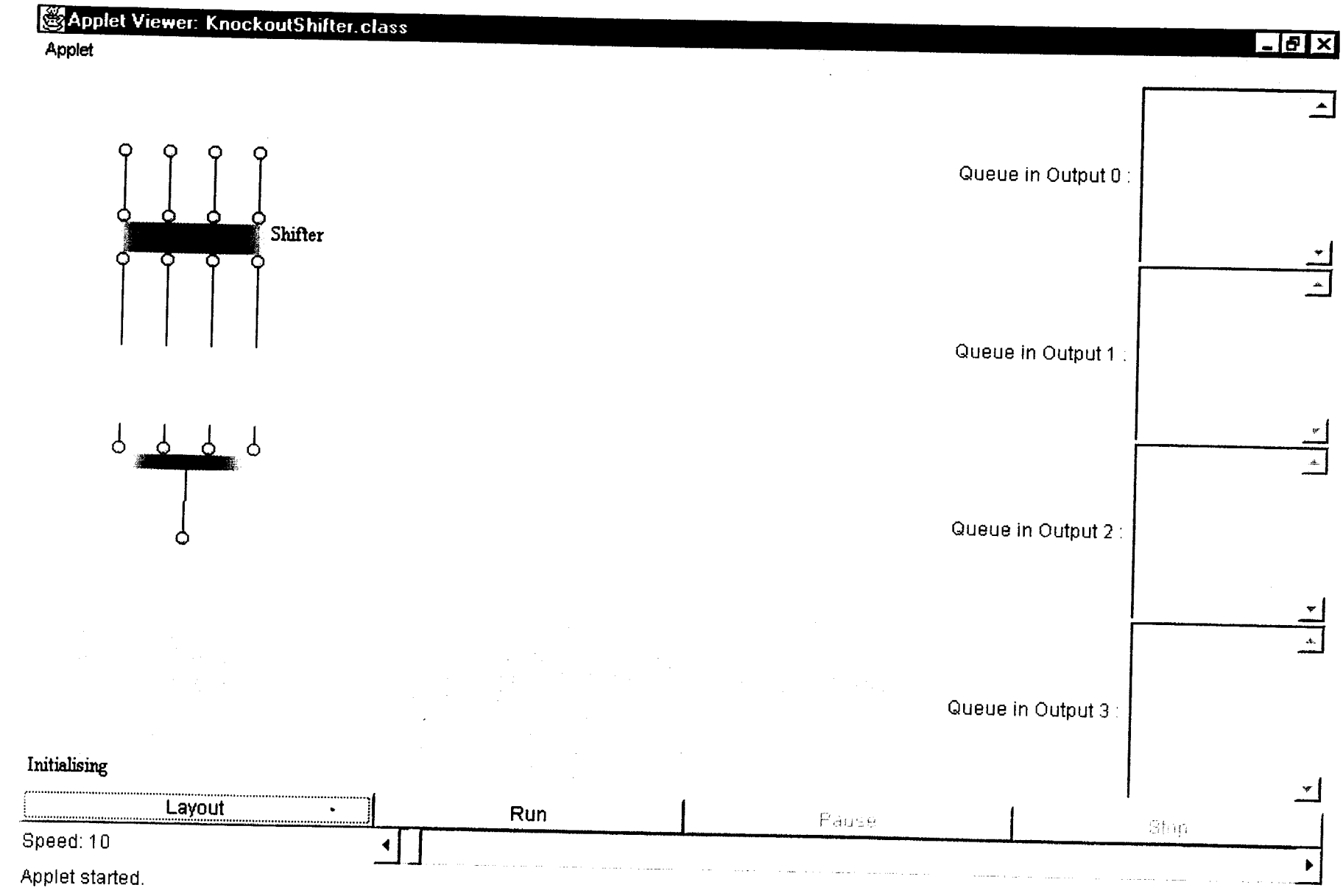
Initialising

Layout | Run | Pause | Stop

Speed: 10

Applet started.

DESIGN OF SHARED BUFFER


Applet Viewer: KnockoutShifter.class
Applet
Shifter
Queue in Output 0 :
Queue in Output 1 :
Queue in Output 2 :
Queue in Output 3 :
Initialising
Layout
Run
Pause
Step
Speed: 10
Applet started.

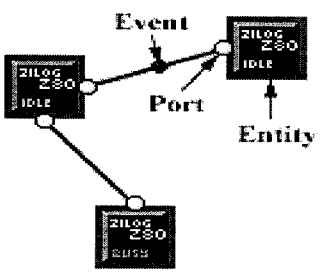
CHAPTER 4

SOFTWARE CONFIGURATION

PROGRAMMING ENVIRONMENT –SOFTWARE CONFIGURATION

4.1 SIMJAVA AS OUR SYSTEM TOOL

Simjava, is a discrete event simulation package for Java. A simjava simulation is a collection of entities each running in its own thread. These entities are connected together by ports and can communicate with each other by sending and receiving event objects. A central system class controls all the threads, advances the simulation time, and delivers the events. The progress of the simulation is recorded through trace messages produced by the entities and saved in a file.



A simulation layout

For example, in the trial program diagram shown follows, we can explain what entities are and how they can perform discrete event simulations: In this diagram. There are 7 entities (Host A, Host B, Host C, Host D, Host E, Host F and Find Friends). The host on the left-hand side is to send message to the hosts on the right hand side. The host

will be selected by the choices shown in the top of this applet window. The bottom part is the control panel for adjusting the speed of the simulation as well as to control the time to start and stop the simulation. This control panel is the standard panel in every simulation.

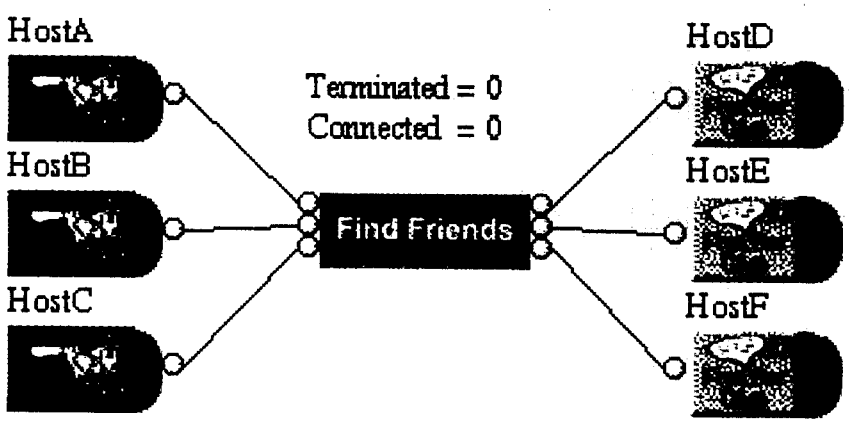
Once the RUN button is pressed, the program will run and we will see an event is sent by the selected host and move forward through the "Find Friends" entities and go to the destination.

Applet Viewer: Trial1.class

Applet

From Host : **Choose Host**

To Host : **Choose Host**



Initialising

Layout | Run | Pause | Skip

Speed: 10

Applet started.

4.2 A SIMPLE ILLUSTRATION

Constructing a simulation involves:

- Coding the behavior of the simulation entities. This is done by extending the standard Sim_entity class and overriding the body () method.
- Adding instances of these entities to the static Sim_system object, using Sim_system.add (entity).
- Linking the entities' ports together, using Sim_system.link_ports().
- And, finally, setting the simulation in motion by calling Sim_system.run()

The following example is a simulation with two entities; a source and a sink. The source schedules 100 messages to the sink, waiting for an acknowledgement and holding for 10 simulation time units between each schedule.

The main file:

```
import eduni.simjava.*;
class Example
{
  public static void main(String args[])
  {
     Sim_system.initialise();
     Sim_system.add(new Source("Sender", 1, Source.SRC_OK));
```

```
    Sim_system.add(new Sink("Receiver", 2, Sink.SINK_OK));
    Sim_system.link_ports("Sender", "out", "Receiver", "in");
    Sim_system.run();
} }
```

The first line imports all the classes in the simjava package, all source files in the simulation should have it at the top.

The main program is very simple and follows the steps laid out above. First the Sim_system object is initialized with the call Sim_system.initialise(), this should be done at the start of every .Then the entities are linked together by the Sim_system.link_ports() call. It links the port called "out" on the "Sender" entity to the port called "in" on the "Receiver" entity. Finally the simulation is set in motion by calling Sim_system.run(), which will exit when there are no more events to process.The source's body() function includes all the most important methods of simjava:

- sim_schedule(Sim_port port, double delay, int tag) sends a message to the entity connected to the port, in delay simulation time units from now, with the given tag.
- sim_wait(Sim_event ev) waits for an event sent using sim_schedule(), (in this case from the Sink entity).
- sim_hold(double t) blocks the entity for t simulation time units.
- sim_trace(int level, String msg) adds the msg to the trace file

4.2.1 COMPILING SIMULATIONS

To compile the program you need to first add the simjava package to your CLASSPATH environment variable so the Java compiler knows where to find it. The classes are found in the classes/ sub-directory off simjava's base directory. After running, the simulation will leave a file called "tracefile" in the current directory, this file contains all the trace lines from the simulation.

4.2.2 RUNTIME METHODS

A runtime method is a method which can be called from the body() method of an entity. By convention, all runtime function names begin with the prefix "sim_", a full list can be found in the reference documentation for the class sim_entity. A few of the methods are given below:

sim_trace(int level, String msg)

This adds the msg to the trace file. The level can be used to control which traces get printed in a simulation run, without having to manually comment out sim_trace() calls in the code.

sim_schedule()

This method comes in three flavors, depending on how you refer to the destination of the event, (the first argument).

- sim_schedule(Sim_port port, ...) - Send the event to the entity attached to the other end of the port, pointed to by port and owned by this entity.
- sim_schedule(String port_name, ...) - Send the event to the entity attached to the other end of the port called port_name owned by this entity.
- sim_schedule(int entity_id, ...) - Send the event to the entity with id number entity_id. Entities are assigned unique id numbers, starting at 0, by the Sim_system as they are added.
- sim_wait(event)

If the parameter event points to a blank Sim_event object, then it is set to the event received and information can be extracted from it.

```
Sim_event ev = new Sim_event();

sim_wait(ev);
```

4.3 FEATURES OF SIMJAVA

4.3.1 Overview of classes: Sim_system

- The Sim_system class has several static methods for monitoring

the simulation, and altering its global flags. (appendix)

- It manages a simjava simulation. Once initialization is complete, it enters a run() loop which runs all entities as far as they can go,

pops the next event off the future event queue, deals with it (which makes the destination entity concerned runnable), and repeats.

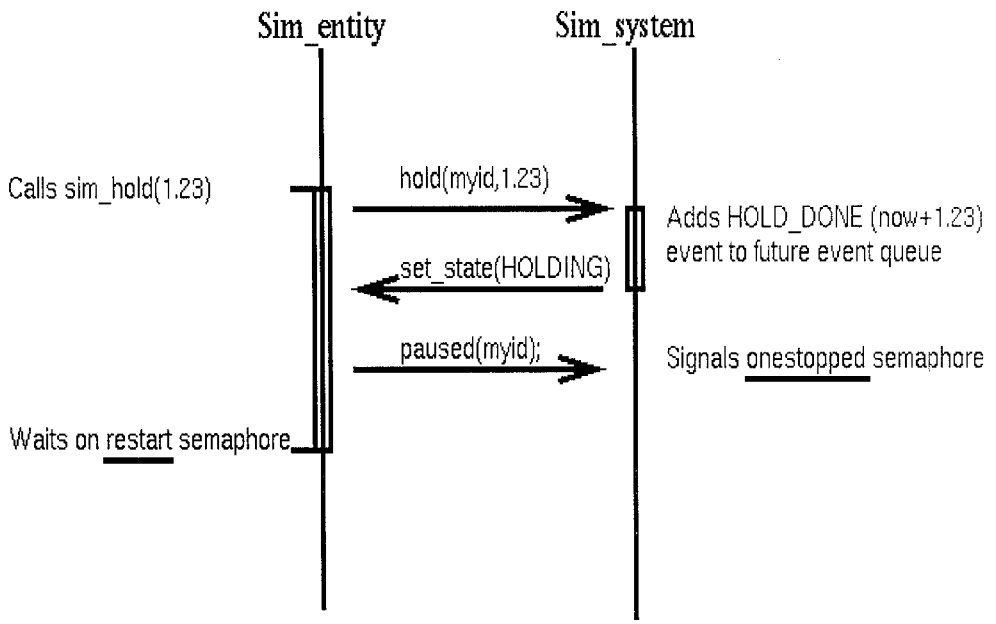
The run() method is implemented as follows:-

```
  static public void run()
{
   run_start();
   while(run_tick())
 {
}
   run_stop();
 }
```

run_tick() runs a single simulation tick. It sets all runnable entities running, waits for them all to grind to a halt, then pops an event off the future queue.

Sim_system is an entirely static class, so doesn't need to be instantiated to be used.

The figure below shows the interactions:-



4.3.2 Sim_entity

The class from which all user simulation entities are derived. Its 'body()' method is called at simulation run time. The basic simulation operations (sim_schedule, sim_wait etc) are methods of Sim_entity. These methods make calls to the central Sim_system class which adds or removes corresponding events from its event queues.

For example, a call to Sim_hold(123.0) calls Sim_system.hold(myid,123.0) which is implemented as follows:

```
static void hold(int src, double delay)
{
  Sim_event e = new
  Sim_event(Sim_event.HOLD_DONE,clock+delay,src);
  future.add(e);
  ((Sim_entity)entities.elementAt(src)).set_state(Sim_entity.HOLDING);
 }
```

This creates a new event (of type HOLD_DONE), with the timestamp of current time + 123.0, adds it to the future event queue, and sets the state of the calling Sim_entity to HOLDING.

Sim_entity.sim_hold() then notifies Sim_system that it's about to stop and waits on its restart semaphore.

Other Public Classes

- Semaphore Basic counting semaphore used for synchronization.
- Evqueue Stores a timestamp ordered queue of events.
- Sim_event Stores events.
- Sim_port Stores a name and the destination entity to which it is attached.
- A Sim_system method connects ports together.
- static public void link_ports(String ent1, String port1,
  String ent2, String port2)

4.4 FEATURES OF SIMANIM

This is a guide to writing applet animations of simulations, using the simanim extension to the simjava simulation package. It deals only with the animation extensions, and does not tell you how to write the simulations themselves.

Animations written with the simanim extension have the following features available:

- Entities and ports are represented by their own user-defined icons.
- Linked ports are joined together by a `wire'.
- Events sent between entities are animated traveling along the wires.
- Text string parameters can be shown next to entities to display information about their internal state.
- The icon representing an entity can be changed during the simulation to represent its internal state.

4.4.1 simanim Design Notes

The classes making up the simanim package are:

Anim_applet Anim_param Param_type_list

Anim_entity Anim_port Sim_anim

Anim_event Param_type

The important classes in the simanim package are Anim_applet and Sim_anim. Anim_entity, Anim_port and Anim_event store animation-specific details for the corresponding simjava classes Sim_entity, Sim_port and Sim_event. Anim_param, Param_type and Param_type_list store displayed parameters and types.

The user's applet derives from Anim_applet, and provides versions of anim_init to create buttons and anim_layout to build the simulation. Anim_applet's init method creates the panel, creates an instance of Sim_anim, creates the control buttons, and calls the user's anim_init().

Sim_anim's animate() method controls the running of the simulation. The three Sim_system methods used are:-

```
Sim_system.run_start();
running = Sim_system.run_tick();
Sim_system.run_stop();
```

4.4.2 The Anim_applet class

Animations are written by extending the Anim_applet class in the simanim package. There are two methods which the subclass needs to override:

anim_layout()

All the code needed to build the simulation should be put in here, it effectively replaces the main() method in stand-alone simulations, and is called before each simulation run begins. Note, there is no need to

call Sim_system.initialise() at the start of this method, or Sim_system.run() at the end, Anim_applet does this for you.

anim_init()

This method is called by Anim_applet during its own initialisation. Any code to add GUI objects to the applet for input should be put in here. Note that you should not override Anim_applet's own init() method.

Anim_applet uses the border layout manager, the ``Center'' I is used for the main animation window so it gets as much space as is available, and the ``South'' for some standard controls. The ``North'' is left empty for the user's GUI components, which can be added by a this.add("North", guiComponent); call from anim_init().

All the source files in your animation must import the two packages eduni.simjava and eduni.simanim in order to compile.

CHAPTER 5

PROJECT IMPLEMENTATION

PROJECT IMPLEMENTATION

We have successfully simulated the working of the Knockout switch using simjava to illustrate its merits over the other conventional switches. The output can be viewed by using a java enabled web browser or applet viewer. The simulation panel consists of several icons Layout, run, pause and stop. The time taken for the simulation to run and output to be obtained is also indicated. The simulation can be run at the desired speed using an icon which can be dragged across.

First we have simulated the working of a knockout switch concentrator. The winner or loser at each stage is decided and after the completion of all stages the winners appear at the corresponding outputs.

Second we show the working of a shared buffer in which the packets arriving are stored in the buffer in a cyclic fashion and they follow the First-in-first-out principle to arrive at the output one after another. At the right hand side of the screen a text box has been created to show the arrival and queuing of packets as they routed one after another to the output.

CHAPTER 6

SOURCE CODE

SOURCE CODE

6.1 CONCENTRATOR.java

```
import java.applet.Applet;
import java.awt.*;
import java.util.Vector;
import eduni.simanim.*;
import eduni.simjava.*;
/ * The ConcentratorInputs class is an input of the concentrator in the
 * Knockout Switch*/
class ConcentratorInputs extends Sim_entity
{
   /* the port that adheres the inputs of the concentrator*/
   private Sim_port in;
   /* the identity number that specifies the inputs */
   private int number;
  /* Constructor that initialize the properties of the inputs of concentrator
    * @param name the name of the concentrator
    * @param number the identity number that specifies the input
    * @param x the x-coordinate of the input should locate
    * @param y the y-coordinate of the input should locate
    */
   public ConcentratorInputs( String name, int number, int x, int y )
   {
        super( name, "dotPort", x, y );
```

```
        this.number = number;
        in = new Sim_port( "conIn"+number, "dotPort", Anim_port.RIGHT, 0 );
        add_port( in );
        add_param( new Anim_param( "", Anim_param.VALUE, "", 0, -5 ) );
    }
    /* The main body method that defines and controls the behaviour of each input*/
    public void body()
    {
        Sim_event ev = new Sim_event();
        sim_schedule( "conIn"+this.number, 0.0, this.number, "P"+this.number );
        sim_trace( 1, "S conIn"+this.number + " P"+this.number );
        sim_hold(0.01);
    }
}

/* The ConcentratorOutputs class is an entity that represents the output of
 * the concentrator. Each output will receive a message from the input, put
 * it into a shifter and go to the output of the Knockout switch. */
class ConcentratorOutputs extends Sim_entity
{
    /* the port that adheres to the output of the concentrator */
    private Sim_port out;
    /* the identity number that specifies the output destination */
    private int number;
    /*Constructor that initializes the properties of each output of the
     * concentrator
```

```
     * @param name the name of the output
     * @param number the identity number of the output
     * @param x the x-coordinate of the output should locate
     * @param y the y-coordinate of the output should locate
     */
    public ConcentratorOutputs( String name, int number, int x, int y )
    {
        super( name, "dotPort", x,  y );
        this.number = number;
        out = new Sim_port( "conOut"+number, "dotPort", Anim_port.LEFT, 0 );
        add_port( out );
        add_param( new Anim_param( "Winner", Anim_param.NAME_VALUE, "", 0, 20 )
);
    }
    /*The main body method that defines and controls the bahaviour of each output of the
concentrator */
    public void body()
    {
        while( true )
        {
            Sim_event ev = new Sim_event();
            sim_get_next(ev);
            sim_trace( 1, "P " + ev.get_data().toS
string() );
            sim_hold(0.01);
        }
```

```
    }
}

/*The Contention Switch class represents the switches in the concentrator. When two
cells go into the contention switch. The switch will assign a winner and  let it go to the
winner port and the other will go to the lose port.
*/
class Contention Switch extends Sim_entity
{
    /*the port that adheres to the input of Contention switch */
    private Sim_port swIn;
    /* the port that adheres to the output of the Contention switch */
    private Sim_port swOut;
   /*the identity number that represent the contention switch*/
    private String number;
   /* the state of the contention switch*/
    private int state;
    /*the number of waiting message for that contention switch*/
    private int wait;
   /* to indicate whether the contention switch should have a double port or not*/
    private boolean doublePorts1;
   /* to indicate whether the contention switch should have only one double Input port and
a single output port */
    private boolean doublePorts2;
   /*a vector to store the messages that reach the contention switch*/
    private Vector numVector = new Vector();
```

```
    /*the CROSS state of the contention switch */
    private static final int CROSS = 0;
   /*the BAR state of the contention switch*/
    private static final int BAR = 1;
   /*the IDLE state of the contention switch*/
    private static final int IDLE = -1;
  /*onstructor that initializes the properties of the Contention switch
    * @param name the name of the contention switch
    * @param number the identity number that specifies the contention switch
    * @param doublePorts to indicate whether the contention switch has a double port
    * @param x the x-coordinate of the contention switch should locate
    * @param y the y-coordinate of the contention switch should locate
    */
  public Contention Switch( String name, String number, boolean doublePorts, int x, int
y )
  {
      super( name, "concentrator", x, y );
      String[] doubleSwitch1 = { "000", "001", "002", "003", "010", "011",
                                 "020", "100", "101", "110", "111", "120",
                                 "130", "121", "200", "210", "211", "220",
                                 "230", "240" };
      String[] doubleSwitch2 = { "310", "320", "330", "340" };
      this.state = IDLE;
      this.number = number;
      this.wait = 0;
      this.numVector.removeAllElements();
```

```
  for( int i = 0; i < 2; i++ )
{
      swIn = new Sim_port( "contentIn"+this.number+""+i,
                           "dotPort", Anim_port.TOP, 6*(i+1) );
      add_port( swIn );
      swOut = new Sim_port( "contentOut"+this.number+""+i,
                            "dotPort", Anim_port.BOTTOM, 6*(i+1) );
      add_port( swOut ); }
 add_param( new Anim_param( "State", Anim_param.STATE, "idle",
                                                          0, 0 ) );


   for( int j = 0; j < doubleSwitch1.length; j++ )
   {
      if( doubleSwitch1[j].equals(this.number) )
      {
            this.doublePorts1 = true;
            break;
      }
      else
            this.doublePorts1 = false;
   }
   for( int k = 0; k < doubleSwitch2.length; k++ )
   {
      if( doubleSwitch2[k].equals(this.number) )
      {
            this.doublePorts2 = true;
```

```
                break;
            }
            else
                this.doublePorts2 = false;
        }
}
 /* The main body method that defines and controls the behaviour of each
 * contention switch */
public void body()
{
    while( true )
    {
        Sim_event ev = new Sim_event();
        this.wait = this.sim_waiting();
        sim_hold( 1.0);
        for( int i = 0; i < this.wait; i++ )
        {
            if( this.wait == 2 )
                competition();
            if( this.wait == 1 )
            {
                ev = new Sim_event();
                sim_get_next(ev);
                winner( ev );
            }
```

```
        }
        sim_hold(0.01);
    }
}

/*a method for two message competed to win */
private void competition()
{
    boolean win = false;
    Sim_event ev = new Sim_event();
    sim_get_next(ev);
    if( Math.random() < 0.5 ) // this event wins
    {
        win = true;
        winner( ev );
    }
    else
        loser( ev );
    sim_get_next(ev);
    if( doublePorts2 == true ) // this switch has two input and one output ports
    {
        if( win == true )
            sim_cancel( new Sim_type_p(ev.get_tag() ), ev );
        else
            winner( ev );
    }
```

```
        if( doublePorts1 == true ) // this switch has two input and output ports
        {
           if( win == true )
                loser( ev );
           else
                winner( ev );
        }
        else
           sim_cancel( new Sim_type_p(ev.get_tag() ), ev );
        win = false;
    }
    /*a method for a winning event to go to the winner port
     * @param ev the waiting event*/
    private void winner( Sim_event ev )
    {
         sim_schedule(  "contentOut"+this.number+"0",  0.0,  ev.get_tag(),
ev.get_data() );
         sim_trace( 1, "S contentOut"+this.number+"0 " + ev.get_data().toString() );
    }
    /* a method for a losing event to go to the loser port
      * @param ev the waiting event */
    private void loser( Sim_event ev )
    {
        sim_schedule( "contentOut"+this.number+"1", 0.0, ev.get_tag(), ev.get_data() );
        sim_trace( 1, "S contentOut"+this.number+"1 " + ev.get_data().toString() );
    }
```

```
}
/* The Concentrator class represents the concentrator component in the Knockout
Switch
 * It will behave as a tournament that each message need to compete for winning so that
they can go to the output of the concentrator. */
public class Concentrator extends Anim_applet
{
  /* to draw the number of entities that involved in the simulation and link them together
*/
  public void anim_layout()
  {
        String[] conArray = { "000", "001", "002", "003", "010",
                              "011", "020", "030", "040", "050", "060",
                              "070", "100","101","110", "111", "120",
                              "121", "130", "140", "150","150",
                              "160", "200", "210", "211", "220",
                              "221","230", "231", "240", "250", "300",
                              "310", "320","330", "340" };
    int gap = 0; // whether there should be a gap between the contention switch
    for( int i = 0; i < conArray.length; i++ )
    {
      if( i != 0 ) gap = 1;
      String number = conArray[i];
      int bit1 = new
Character(number.charAt(0)).getNumericValue(number.charAt(0));
```

```
        int bit2 = new
Character(number.charAt(1)).getNumericValue(number.charAt(1));
        int bit3 = new
Character(number.charAt(2)).getNumericValue(number.charAt(2));
        Sim_system.add( new Contention Switch( "ConSwitch"+number, number,
false, 50+140*bit1+30*bit3, 50+40*bit2+40*bit1 ));
      }
      gap = 0;

      for( int j = 0; j < 8; j++ )
      {
        if( j != 0 ) gap = 1;
        Sim_system.add( new ConcentratorInputs( "ConInput"+j, j,
                                                50+20*j-10*(j/2), 30 ) );
        if( j < 4 )
            Sim_system.add( new ConcentratorOutputs( "ConOutput"+j, j,
                                                56+140*j, 400 ) );
      }

      //link the inputs with the switches together
      for( int i = 0; i < 8; i++ )
      {
        if( i%2 == 0 )  // connect to the port 0 of each switch
            Sim_system.link_ports( "ConInput"+i, "conIn"+i,
                                   "ConSwitch00"+i/2, "contentIn00"+i/2+"0" );
        else
```

```
            Sim_system.link_ports( "ConInput"+i, "conIn"+i,
                                   "ConSwitch00"+i/2, "contentIn00"+i/2+"1" );
        }

        //link the outputs with the switches
        Sim_system.link_ports( "ConOutput0", "conOut0", "ConSwitch070",
                               "contentOut0700" );
        Sim_system.link_ports( "ConOutput1", "conOut1", "ConSwitch160",
                               "contentOut1600" );
        Sim_system.link_ports( "ConOutput2", "conOut2", "ConSwitch250",
                               "contentOut2500" );
        Sim_system.link_ports( "ConOutput3", "conOut3", "ConSwitch340",
                               "contentOut3400" );

        //link the switches together
        //1. Link the switches in the same block 0
        Sim_system.link_ports( "ConSwitch000", "contentOut0000",
"ConSwitch010","contentIn0100" );
        Sim_system.link_ports( "ConSwitch010", "contentOut0100",
"ConSwitch020","contentIn0200" );
        Sim_system.link_ports( "ConSwitch020", "contentOut0200",
"ConSwitch030","contentIn0300" );
        Sim_system.link_ports( "ConSwitch030", "contentOut0300",
"ConSwitch040","contentIn0400" );
        Sim_system.link_ports( "ConSwitch040", "contentOut0400",
"ConSwitch050","contentIn0500" );
```

```
        Sim_system.link_ports(          "ConSwitch050",          "contentOut0500",
"ConSwitch060","contentIn0600" );
        Sim_system.link_ports(          "ConSwitch060",          "contentOut0600",
"ConSwitch070","contentIn0700" );
        Sim_system.link_ports(          "ConSwitch001",          "contentOut0010",
"ConSwitch010","contentIn0101" );
        Sim_system.link_ports(          "ConSwitch002",          "contentOut0020",
"ConSwitch011","contentIn0110" );
        Sim_system.link_ports(          "ConSwitch003",          "contentOut0030",
"ConSwitch011","contentIn0111" );
        Sim_system.link_ports(          "ConSwitch011",          "contentOut0110",
"ConSwitch020","contentIn0201" );
        //2. Link the switches in the same block 1
        Sim_system.link_ports(          "ConSwitch100",          "contentOut1000",
"ConSwitch110","contentIn1100" );
        Sim_system.link_ports(          "ConSwitch110",          "contentOut1100",
"ConSwitch120","contentIn1200" );
        Sim_system.link_ports(          "ConSwitch120",          "contentOut1200",
"ConSwitch130","contentIn1300" );
        Sim_system.link_ports(          "ConSwitch130",          "contentOut1300",
"ConSwitch140","contentIn1400" );
        Sim_system.link_ports(          "ConSwitch140",          "contentOut1400",
"ConSwitch150","contentIn1500" );
        Sim_system.link_ports(          "ConSwitch150",          "contentOut1500",
"ConSwitch160","contentIn1600" );
```

```
        Sim_system.link_ports(          "ConSwitch101",          "contentOut1010",
"ConSwitch110","contentIn1101" );
        Sim_system.link_ports(          "ConSwitch111",          "contentOut1110",
"ConSwitch120","contentIn1201" );
        Sim_system.link_ports(          "ConSwitch121",          "contentOut1210",
"ConSwitch130","contentIn1301" );
        //3.Link the switches in the same block 2
        Sim_system.link_ports(          "ConSwitch200",          "contentOut2000",
"ConSwitch210","contentIn2100" );
        Sim_system.link_ports(          "ConSwitch210",          "contentOut2100",
"ConSwitch220","contentIn2200" );
        Sim_system.link_ports(          "ConSwitch220",          "contentOut2200",
"ConSwitch230","contentIn2300" );
        Sim_system.link_ports(          "ConSwitch230",          "contentOut2300",
"ConSwitch240","contentIn2400" );
        Sim_system.link_ports(          "ConSwitch240",          "contentOut2400",
"ConSwitch250","contentIn2500" );
        Sim_system.link_ports(          "ConSwitch211",          "contentOut2110",
"ConSwitch220","contentIn2201" );
        Sim_system.link_ports(          "ConSwitch221",          "contentOut2210",
"ConSwitch230","contentIn2301" );
        Sim_system.link_ports(          "ConSwitch231",          "contentOut2310",
"ConSwitch240","contentIn2401" );
        //4.Link the switch in the same block 3
        Sim_system.link_ports(          "ConSwitch300",          "contentOut3000",
"ConSwitch310","contentIn3100" );
```

```
Sim_system.link_ports( "ConSwitch310", "contentOut3100",
"ConSwitch320","contentIn3200" );
Sim_system.link_ports( "ConSwitch320", "contentOut3200",
"ConSwitch330","contentIn3300" );
Sim_system.link_ports( "ConSwitch330", "contentOut3300",
"ConSwitch340","contentIn3400" );
//5.Link Block 0 to Block 1
Sim_system.link_ports( "ConSwitch000", "contentOut0001",
"ConSwitch100","contentIn1000" );
Sim_system.link_ports( "ConSwitch001", "contentOut0011",
"ConSwitch100","contentIn1001" );
Sim_system.link_ports( "ConSwitch002", "contentOut0021",
"ConSwitch101","contentIn1010" );
Sim_system.link_ports( "ConSwitch003", "contentOut0031",
"ConSwitch101","contentIn1011" );
Sim_system.link_ports( "ConSwitch010", "contentOut0101",
"ConSwitch111","contentIn1110" );
Sim_system.link_ports( "ConSwitch011", "contentOut0111",
"ConSwitch111","contentIn1111" );
Sim_system.link_ports( "ConSwitch020", "contentOut0201",
"ConSwitch121","contentIn1210" );
//6.Link Block 1 to Block 2
Sim_system.link_ports( "ConSwitch100", "contentOut1001",
"ConSwitch200","contentIn2000" );
Sim_system.link_ports( "ConSwitch101", "contentOut1011",
"ConSwitch200","contentIn2001" );
```

```
Sim_system.link_ports( "ConSwitch110", "contentOut1101",
"ConSwitch210","contentIn2101" );
Sim_system.link_ports( "ConSwitch111", "contentOut1111",
"ConSwitch211","contentIn2110" );
Sim_system.link_ports( "ConSwitch120", "contentOut1201",
"ConSwitch221","contentIn2210" );
Sim_system.link_ports( "ConSwitch130", "contentOut1301",
"ConSwitch231","contentIn2310" );
        //7.Link Block 2 to Block 3
Sim_system.link_ports( "ConSwitch200", "contentOut2001", "ConSwitch300",
"contentIn3000" );
Sim_system.link_ports( "ConSwitch210", "contentOut2101",
"ConSwitch310","contentIn3101" );
Sim_system.link_ports( "ConSwitch220", "contentOut2201",
"ConSwitch320","contentIn3201" );
Sim_system.link_ports( "ConSwitch230", "contentOut2301",
"ConSwitch330","contentIn3301" );
Sim_system.link_ports( "ConSwitch240", "contentOut2401",
"ConSwitch340","contentIn3401" );
   }
}
```

6.2 SHIFTER.java

```
import java.applet.Applet;
import java.awt.*;
import java.util.Vector;
import eduni.simanim.*;
import eduni.simjava.*;
/* The Shifter Input class is an entity which represents the output directly from the
concentrator of the Knockout Switch. It will transfer the message through the shifter into
the buffer. */
class ShifterInput extends Sim_entity
{
   /* the port that adheres to the input of this entity */
   private Sim_port in;
   /* the identity number that specifies the input */
   private int number;
   /* to indicate whether the input has to send message */
   private boolean selected;
   /* Constructor that initializes the properties of each input
    * @param name the name of the input
    * @param selected to indicate whether the input has to send message
    * @param number the identity number of this input
    * @param x the x-coordinate of the input should locate
    * @param y the y-coordinate of the input should locate
    */
```

```
public Shifter Input( String name, boolean selected, int number, int x, int y )
  {     super( name, "port", x, y );
        this.number = number;
        in = new Sim_port( "in"+this.number , "dotPort",
                                Anim_port.BOTTOM, 4 );
        add_port( in );
        add_param( new Anim_param( "", Anim_param.VALUE, "", 0, -10 ) );
  }
  /*The main body method that defines and controls the bahaviour of each
     input*/
  public void body()
  {
        while( true )
        {
           if( Math.random() < 0.5 )
           {
                Sim_event ev = new Sim_event();
                int dest = (int)(Math.random()*10);
                sim_schedule( "in"+this.number, 0.0, dest );
                sim_trace( 1, "P P"+dest );
                sim_trace( 1, "S in"+this.number + " P" + dest );
           }
           else
                sim_trace( 1, "P n" );
           sim_hold( 0.3);
        }  }  }
```

```
/* The Shifter Output class is an entity which represents the output after
   passing through the buffers of the Knockout Switch. The output from this
   will be the exact output of the Knockout Switch.*/
class ShifterOutput extends Sim_entity
{
   /*the port that adheres to the output of this entity*/
   private Sim_port out;
   /* the identity number that specifies the behaviour of each output */
   private int number;
   /* Constructor that initialize the behaviour of each output
    * @param name the name of the output
    * @param number the identity number of each output
    * @param x the x-coordinate of the output should locate
    * @param y the y-coordinate of the output should locate
    */
   public ShifterOutput( String name, int x, int y )
   {
        super( name, "port", x, y );
        this.number = number;
        out = new Sim_port( "out", "dotPort", Anim_port.TOP, 4 );
        add_port(out);
        add_param( new Anim_param( "Output", Anim_param.VALUE, "", 0, 0 ) );
   }
   /* The main body method that defines and controls the bahaviour of each
      output */
   public void body()
```

```
    {
        while( true )
        {
            Sim_event ev = new Sim_event();
            sim_get_next(ev);
            //sim_hold( Double.valueOf(ev.get_data().toString()).doubleValue() );
            sim_trace( 1, "P P"+ev.get_tag() );
        } } }
/*The Shifter class represents a shifter in the Knockout switch. It will put the
   message on the input of the buffers.*/
class Shifter extends Sim_entity
{
   /* the port that adheres to the input of the shifter */
   private Sim_port shiftIn
   /*the port that adheres to the output of the shifter */
   private Sim_port shiftOut;
   /* Constructor that initializes the properties of the shifter
    * @param name the name of the shifter
    * @param x the x-coordinate of the shifter
    * @param y the y-coordinate of the shifter */
  public Shifter( String name, int x, int y )
  {
        super( name, "shifter", x, y );
        for( int i = 0; i < 4; i++ )
        {
           shiftIn = new Sim_port( "shiftIn"+i, "port", Anim_port.TOP,
```

```
                                                    4+28*i );
        shiftOut = new Sim_port( "shiftOut"+i, "port",
                                        Anim_port.BOTTOM, 4+28*i );
        add_port( shiftIn );
        add_port( shiftOut );
        add_param( new Anim_param( "Shifter", Anim_param.NAME, "",
                                                        100, 10 ) );
    } }
  /* The main body method which defines and controls the behaviour of
     the Shifter */
public void body()
{
    int num = 0;
    while( true )
  {
      Sim_event ev = new Sim_event();
      int wait = sim_waiting();
      double time = 0.1;
      for( int i = 0; i < wait; i++ )
      {
          sim_get_next(ev);
          if( num > 3 ) num = 0;
          sim_schedule( "shiftOut"+num, 0.0, ev.get_tag(),
                          Double.toString( time+=0.1) );
          sim_trace( 1, "S shiftOut"+num + " P" + ev.get_tag() );
          num++;
```

```
            }
            sim_hold(0.3);
        } } }
/* The Buffer class represents the buffers which are directly below the output of the
shifter in the Knockout Switch. The message will be outputted according to First-In-First-
Out basis. * /
class Buffer extends Sim_entity
{
    /* the port which adheres to the input of the Buffer */
    private Sim_port bufferIn;
   /* the port which adheres to the output of the Buffer */
    private Sim_port bufferOut;
   /*the identity number that specifies the buffer */
    private int number;

    /* the number of events waiting in this buffer */
    private int wait;
    /*a vector to store the waiting events */
    private Vector waitQueue = new Vector();
    private Sim_event event;
    /* Constructor that initializes the properties of each buffer
     * @param name the name of the buffer
     * @param number the identity number of that buffer
     * @param x the x-coordinate of the buffer should lcoate
     * @param y the y-coordinate of the buffer should locate */
    public Buffer( String name, int number, int x, int y )
```

```
{     super( name, "buffer0", x, y );
      this.number = number;
      bufferIn = new Sim_port( "bufferIn"+this.number, "dotPort",
                                          Anim_port.TOP,8);
     bufferOut = new Sim_port( "bufferOut"+this.number, "dotPort",
                                          Anim_port.BOTTOM, 8 );
      add_port( bufferIn );
      add_port( bufferOut );


add_param( new Anim_param( "State", Anim_param.STATE,
                                          "empty", 0, 0 ) );
  }
  /* The main body method which defines and controls the behaviour of each buffer */
  public void body()
  {
      //      Sim_event ev = new Sim_event();
      int frequency = 0;
      int dest = 0;
      while( true )
      {
         Sim_event ev = new Sim_event();
         this.wait = this.sim_waiting();
         sim_get_next( ev );
         changeState( ++this.wait, ev.get_tag(), true );
         System.out.println( this.wait + "Waiting in queue" );
```

```
            dest = this.number;
            sim_hold(Double.valueOf(ev.get_data().toString()).doubleValue());
            sim_schedule( "bufferOut"+dest,0.0, ev.get_tag(), ev.get_data() );
            //this make step by step the event going

            changeState( --this.wait, ev.get_tag(), false );
            System.out.println( "One go and remaining" + this.wait );
            sim_trace( 1, "S bufferOut"+dest + " P" + ev.get_tag() );                    }
}
        /*change the status of the buffer when an event has been in queue
    * @param wait the number of events waiting
    * @param dest the destination of the event will be sent
    * @param add whether the message should be added or deleted */
    private void changeState( int wait, int dest, boolean add )
    {     //If decided to not to deleted, remove the boolean parameter!
          if( wait == 1 )
          { changeText( this.number, dest, add );
             sim_trace( 1, "P one" );
          }
          if( wait == 2 )
          { changeText( this.number, dest, add );
             sim_trace( 1, "P two" );
          }
          if( wait == 3 )
          { changeText( this.number, dest, add );
             sim_trace( 1, "P three" );
```

```
    }
    if( wait == 4 )
    { changeText( this.number, dest, add );
      sim_trace( 1, "P four" );
    }
    if( wait == 0 )
    { changeText( this.number, dest, add );
      sim_trace( 1, "P empty" );
    } }
/* add or delete the event into TextArea
 * @param num the TextArea identity number,
 * @param dest the destination that it will go
 * @param addText true if add the text, false if delete the text */
 private void changeText( int num, int dest, boolean addText )
 {QueueHandler q = new QueueHandler( num );
      if( addText == true )
        q.addText( "P"+dest);
 }}


class KnockOutput extends Sim_entity
{ private Sim_port in;
  private Sim_port out;
  public KnockOutput( String name, int x, int y )
  {   super( name, "knockoutput", x, y );
      for( int i = 0; i < 4; i++ )2
```

```
        {
            in = new Sim_port( "outputIn"+i, "port", Anim_port.TOP, 4+28*i );
            add_port( in );
        }
        out = new Sim_port( "outputOut", "dotport", Anim_port.BOTTOM,
                                                                    50 );
        add_port( out );
    }
    public void body()
    {   while( true )
        {  Sim_event ev = new Sim_event();
           sim_get_next(ev);
           //sim_hold( 0.1 );
           sim_schedule( "outputOut", 0.0, ev.get_tag(), ev.get_data() );
           sim_trace( 1, "S outputOut P" + ev.get_tag() );
           sim_hold( 0.1 );
        }  }  }
/* The KnockoutShifter class is the microscopic view of the shifter-to-buffer
   component of the Knockout Switch. Each switch contains one shifter and
   several buffers. Here we use 4 inputs and 4 outputs for the shifters and
   buffers which are resulted from the 8 to 4 concentrator of the Knockout
   Switch. The output from the buffer will be the actual output of the
   Knockout Switch. */
public class KnockoutShifter extends Anim_applet
{
```

```
TextArea t1 = new TextArea( "", 10, 10,      TextArea.SCROLLBARS_VERTICAL_ONLY
);
TextArea t2 = new TextArea( "", 10, 10, TextArea.SCROLLBARS_VERTICAL_ONLY );
TextArea t3 = new TextArea( "", 10, 10, TextArea.SCROLLBARS_VERTICAL_ONLY );
TextArea t4 = new TextArea( "", 10, 10, TextArea.SCROLLBARS_VERTICAL_ONLY );
  /* to draw the panel of the simulation*/
  public void anim_init()
  {
       Panel outputPanel = new Panel();
       outputPanel.setLayout( new GridLayout( 4, 2 ) );
       outputPanel.add( new Label( "Queue in Output 0 : ",Label.RIGHT));
       t1.setEditable( false );
       outputPanel.add( t1 );
       QueueHandler handler1 = new QueueHandler( 0 );
       handler1.setTextArea( t1 );
       outputPanel.add( new Label( "Queue in Output 1 : " , Label.RIGHT));
       t2.setEditable( false );
       outputPanel.add( t2 );
       QueueHandler handler2 = new QueueHandler( 1 );
       handler2.setTextArea( t2 );
       outputPanel.add( new Label( "Queue in Output 2 : " , Label.RIGHT));
       t3.setEditable( false );
       outputPanel.add( t3 );
       QueueHandler handler3 = new QueueHandler( 2 );
       handler3.setTextArea( t3 );
       t4.setEditable( false );
```

```
        outputPanel.add( new Label( "Queue in Output 3 : ",Label.RIGHT));
        outputPanel.add( t4 );
        QueueHandler handler4 = new QueueHandler( 3 );
        handler4.setTextArea( t4 );
        this.add( "East", outputPanel );
}
/*to draw out the entities that will be performed in the simulation and
  link them in between.*/
public void anim_layout()
{   t1.setText("");
        t2.setText("");
        t3.setText("");
        t4.setText("");
        for( int i = 0; i < 4; i++ )
        {   if( Math.random() < 0.5 )
              Sim_system.add( new ShifterInput( "Input"+i, true, i, 54+28*i,
                                                                    50 ) );
            else
                Sim_system.add( new ShifterInput( "Input"+i, false, i, 54+28*i,
                                                                    50 ) );
              Sim_system.add( new Buffer( "Buffer"+i, i, 50+28*i, 180 ) );
        }
        Sim_system.add( new Shifter( "Shifter", 50, 100 ) );
        Sim_system.add( new KnockOutput( "KnockOutput", 50, 250 ));
        Sim_system.add( new ShifterOutput( "Output", 95, 300 ) );
        for( int j = 0; j < 4; j++ )
```

```
        {  Sim_system.link_ports( "Input"+j, "in"+j, "Shifter", "shiftIn"+j );
           Sim_system.link_ports( "Shifter", "shiftOut"+j, "Buffer"+j,
                                                                "bufferIn"+j );
           Sim_system.link_ports( "Buffer"+j, "bufferOut"+j, "KnockOutput",
                                                                "outputIn"+j );
        }
Sim_system.link_ports( "KnockOutput","outputOut","Output","out" );
   }}
```

6.3 QUEUE HANDLER.java

```
import java.awt.*;
import java.util.*;
/* A Queue Handler class represents the queue in buffer of the Knockout switch. It helps
to modify the queue textarea if an event comes so that the use knows which messages
are in the queue of which buffers. */
public class QueueHandler
{
  /* stringbuffer stores the string of messages in queue  */
  private StringBuffer buffer;
  /* the vector which stores the textarea object */
  private static Vector fieldVector = new Vector();
    /* the textarea that the string will be input in */
  private static TextArea t;
  /* the identity number of the textarea that it represents */
  private int number;
  /* Constructor that assigns the specified textarea and the stringbuffer  */
  public QueueHandler( int number )
  {
      this.number = number;
      this.buffer = new StringBuffer();
}
/* add the text into the textarea
```

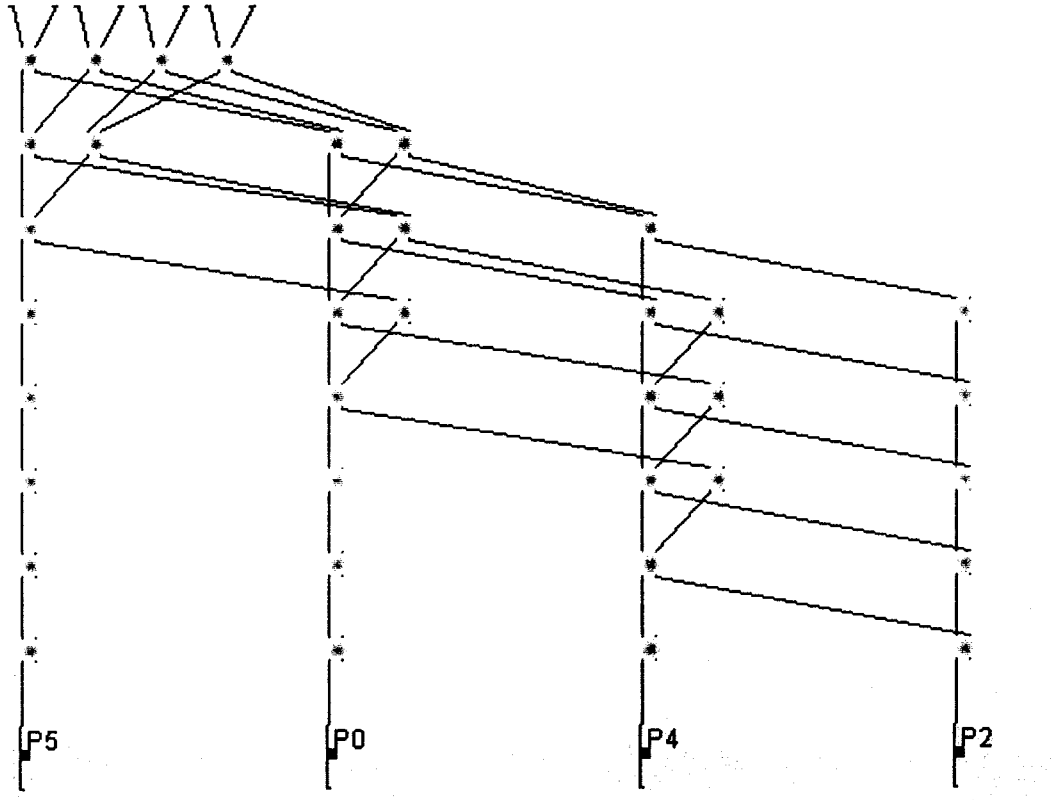
```
 * @param cellName the name of the event that adds into the textarea */
 public void addText( String cellName )
{
      this.getTextArea().append( cellName + " " );
}
/*add the TextArea object into the vector
 * @param area the textarea */
public void setTextArea( TextArea area )
{
      fieldVector.addElement( area );
}
/* get the specified textarea with the specified identity number
 * @return the specified textarea */
private TextArea getTextArea()
{     return (TextArea)fieldVector.elementAt( this.number );
} }
```

CHAPTER 7

SAMPLE OUTPUT

Applet Viewer: Concentrator.class

Applet

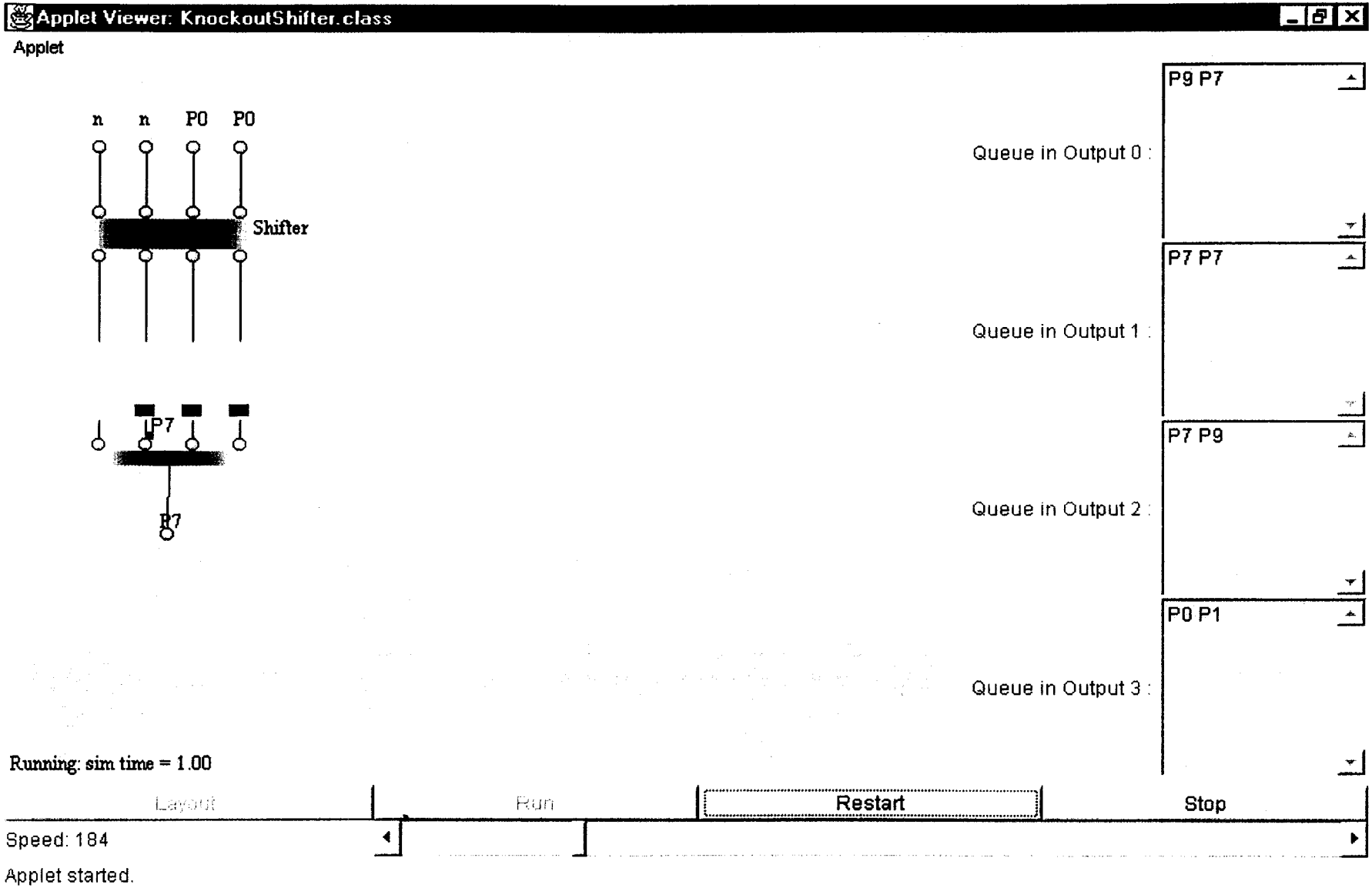


Running: sim time = 9.08

Layout • Run Restart Stop

Speed: 182

Applet started.


Applet Viewer: KnockoutShifter.class
Applet
n
n
P0
P0
Shifter
P7
P7
P9 P7
Queue in Output 0 :
P7 P7
Queue in Output 1 :
P7 P9
Queue in Output 2 :
P0 P1
Queue in Output 3 :
Running: sim time = 1.00
Layout
Run
Restart
Stop
Speed: 184
Applet started.