

PERFSIM: A Performance Simulator using VHDL

Guru Prasad*
University of Cincinnati

Peter Frey†
Fachhochschule Ulm

Harold Carter‡
University of Cincinnati

Abstract

A general queuing network-based simulation system useful for analyzing the performance and resource loading of computing systems has been developed which uses VHDL as the simulation engine. The system to be simulated is described as an extended queuing network in the simple PERFSIM language. A program, written in C++, converts the description into VHDL code which is then analyzed and executed to simulate the system. This paper describes the PERFSIM simulator and input language.

Introduction

During the early stages of the development of computer-based systems simulation of possible architectural configurations is often necessary to select the most desirable organization for further development. Usually, some form of token-based queuing network simulation is used where the architecture is modeled as a queuing network and typical workloads are modeled as flows of tokens [1]. PERFSIM has been developed to be such a simulator [2].

PERFSIM takes a file, written in the PERFSIM language, which describes both the queuing network and the workload as tokens, converts the description into VHDL, and executes the VHDL simulation. The simulation results include both trace data and data statistics.

*University of Cincinnati, Department of Electrical and Computer Engineering, ML 30, Cincinnati Ohio 45221-0030

†Fachhochschule Ulm, Prittwitzstraße 10, Postfach 38 60, 89075 Ulm/Donau, Germany

‡University of Cincinnati, Department of Electrical and Computer Engineering, ML 30, Cincinnati Ohio 45221-0030

PERFSIM queuing network components consist of four components: a source of tokens, a queue, a server, and a sink. Inputs and outputs of each of these components can be connected together in a variety of ways to represent the computer architecture and execution characteristics. The token source component establishes the initial contents of a token and the rate of token generation based on a variety of random distributions. A queue is a buffer (normally FIFO) which can have multiple inputs but only a single output. A server is a very general component which can accept multiple inputs, perform a variety of operations on token data, and multiplexor output tokens with a variety of sequencing options. A sink is a single input component which destroys tokens.

Some unique features of PERFSIM include the generality of server operation, the use of “probes” to sample and report time and other token data activity, and the use of VHDL as the internal simulation language. Over the past year, PERFSIM has been used in several projects including the analysis of parallel computing architectures for real-time execution of an Air Force counter-measure simulation engine [3].

PERFSIM Description

Although one can create queuing network simulation models for computing systems directly in VHDL, we chose instead to create a much more efficient language, called the PERFSIM language, and compile descriptions in this language to VHDL. Thus, we use VHDL as an intermediate format.

A system is defined in PERFSIM as a network of sources, sinks, queues, and servers. Sources,

queus, and sinks operate as expected. The server component is a very general module which holds tokens for a user-defined period of time representing modeled service delays. The general module also can extract and modify any of three value fields in a token, and a variety of input and output selection policies can be defined for multiple input and output interfaces.

A simple PERFSIM input description is shown in Figure 1. This description is for a simple open queue-server model with a single source randomly emitting tokens every 100 nsecs (uniform) average, and a sink to remove tokens from the system. The server (GENMOD module in Figure 1) has a constant delay of 100 nsecs. The last section (called “TOP”) of the input is the network description. The queuing network diagram for this input is shown in Figure 2. Capitalized words are PERFSIM keywords while small-case words are variables, module names, or time units.

PERFSIM first reads the input and parses it using a parser created by the GNU BISON program. Normally, each input PERFSIM module produces a VHDL entity and architecture although component instantiation is used if the same PERFSIM module is instantiated more than once in the NET or TOP modules. The VHDL descriptions are then compiled (using the Vantage Analysis Systems, Inc., VHDL simulator) and elaborated along with two VHDL packages that provide math functions and main VHDL entities for the entire VHDL simulation. The simulations are then executed producing a standard summary simulation output report as shown for one general module called *io* in Figure 3. In the example shown, $11 - 9 = 2$ tokens were in the system when simulation terminated. “Time spent” refers to length of time a token is in the system. “Amount of tokens between points” is the number of tokens observed between any input and output of the module.

PERFSIM Details

Token Passing between Modules

```

SOURCE src
  OUTPUTS outx;
  MECH RAND 100 ns;
  OVERFLOW;
END SOURCE;

QUEUE que
  INPUTS inx;
  OUTPUTS outx;
  LENGTH 20;
  TYPE FIFO;
END QUEUE;

GENMOD gen
  INPUTS inx;
  OUTPUTS outx;
  DELAY 100 ns;
END GENMOD;

SINK snk
  INPUT inx;
END SINK;

TOP
  src(a);
  que(a,b);
  gen(b,c);
  snk(c);
END TOP

```

Figure 1: Simple PERFSIM input example.

Tokens flow between any two modules using a well established protocol. Figure 4 depicts the protocol at an interface between two modules. *sndr*(sender) refers to the output interface of a module and *rcvr*(receiver) refers to the input interface of a module. Further, *inpt*(input) and *oupt*(output) determine the direction of flow for the control signal. *sndr.inpt* refers to the signal coming into the output interface of a module. Thus, at an interface between two modules, the *sndr.oupt* of the sender is connected to the *rcvr.inpt* of the receiver, and *sndr.inpt* of the sender is connected to the *rcvr.oupt* of

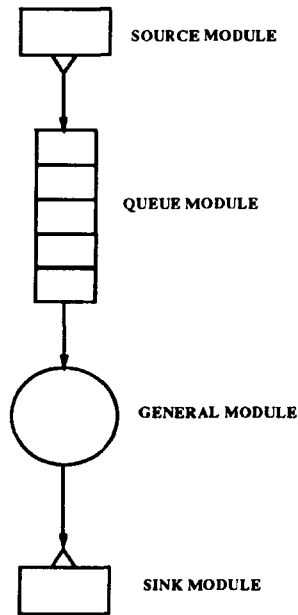


Figure 2: Queuing network represented by PERFSIM input in Figure 1.

the receiver. `sndr.oupt` and `rcvr.inpt` can take on values `tok_avai` (which means a token is available) and `tok_sent` (which means a token has just been sent). Similarly, the `sndr.inpt` and `rcvr.oupt` can take values `rdy` (ready) and `ack` (acknowledge).

Initially, the hand-shaking protocol is as shown in Figure 4(A) where `sndr.oupt` has value `tok_sent` and `rcvr.oupt` has `rdy`. When a token is available at the senders output, it sets the `sndr.oupt` value to `tok_avai` (Figure 4(B)). The receiver which was waiting for this signal now sets its `rcvr.oupt` value to `ack` (Figure 4(C)) after which the sender then sets its `sndr.oupt` value `tok_sent` Figure 4(D).

The receiver module of the token then updates (optionally) the user defined fields of the token sets its `rcvr.oupt` signal to `rdy`. The module can then receive another token. This protocol is followed at all of the interfaces and ensures only one token is transferred at the interface at a time and no token is lost by overwriting.

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X      SUMMARY REPORT      X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXX io XXXXXXXXXXXXXXX

TOTAL NUMBER OF TOKENS ARRIVING = 11
TOTAL NUMBER of TOKENS DEPARTING = 9
TOTAL TIME SPENT = 4305 NS
MAXIMUM TIME SPENT = 524 NS
MINIMUM TIME SPENT = 120 NS
AVERAGE TIME TO PROCESS A TOKEN = 270.6 NS
MAXIMUM AMOUNT OF TOKENS BETWEEN POINTS = 2
MINIMUM AMOUNT OF TOKENS BETWEEN POINTS = 1
AVERAGE AMOUNT OF TOKENS BETWEEN POINTS = 1.1
UTILIZATION = 99.7

```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Figure 3: Example output from PERFSIM

General Module Structure

The structure of the general module is shown in Figure 5. All of the functions of a general module are implemented as a VHDL process. The figure shows the modules with a single input and output. In general they can have more inputs and outputs, in which case it would have a pair of interface signals for each input and output.

0.0.1 Queue module

A queue module is realized by three VHDL processes as shown in Figure 6. The first VHDL process inserts a token in the queue, the second process deletes a token from the queue, and the third synchronizes token insertions and deletions. The queue itself is realized as an array of tokens. A pointer is maintained in the process for insertion. The pointer always points to the next vacant position in the queue. Similarly, the deletion process maintains a pointer that points to the next token to be output.

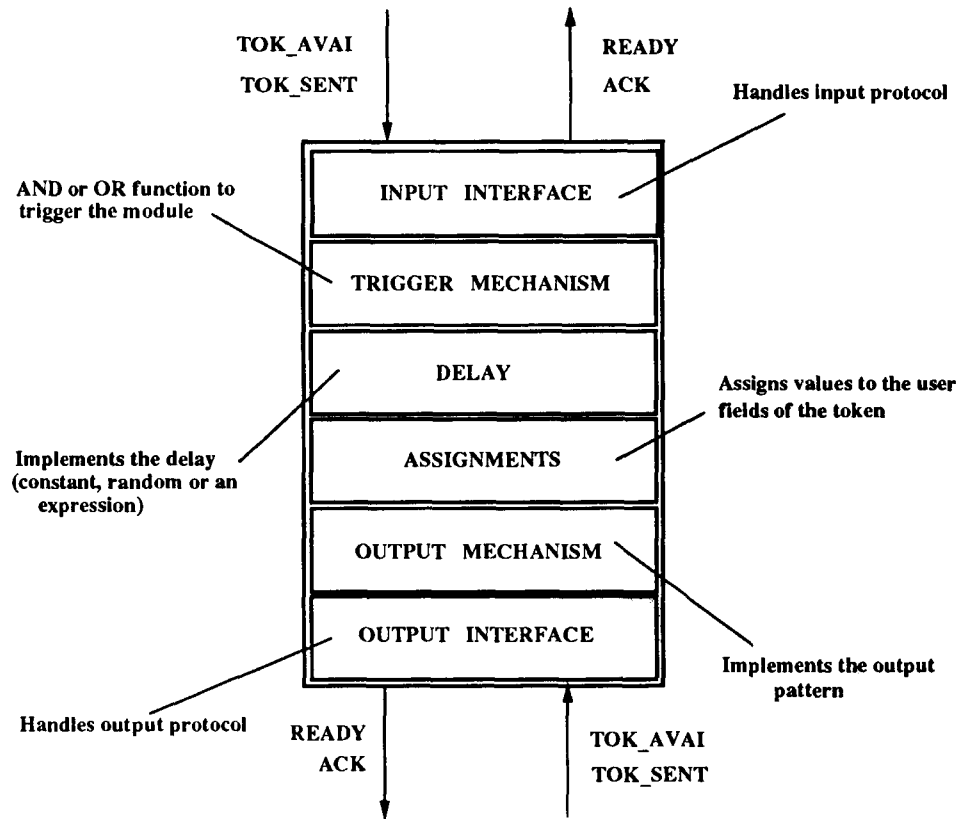


Figure 5: Anatomy of the General module.

Module Synchronization

The synchronization between insertion and deletion processes in a queue module is achieved by the following locking mechanism.

The lock is maintained by a queue synchronizing mechanism. When a token arrives at the input of a module it requests a lock from the synchronizing process. If the lock is not already held by the deletion process, the synchronizing process grants the lock to the insertion process which inserts the token in the queue. After the insertion process is done, it requests the lock to be released by the synchronizing process.

If the lock is held by the deletion process when the insertion process requests it, the insertion process will wait until the lock is available. Once the lock is released by the deletion process the insertion process proceeds by locking.

Report Generation

In PERFSIM, the user specifies the points that are to be monitored during simulation to produce a report. Monitoring is achieved using the *probe* specification in the model description. A summary report and an optional trace report are produced.

A VHDL process is created for each probe of type INTO inserted in the PERFSIM input, while two VHDL processes are created for each probe of type BETWEEN. These processes are placed in the topmost module of the VHDL description. Each VHDL probe process monitors the interface specified by the user between two modules in the model. The process monitoring a INTO probe counts the number of tokens appearing at the interface and maintains a history of the minimum and maximum times between tokens.

A BETWEEN probe produces a statistical report of the time taken by tokens to pass from

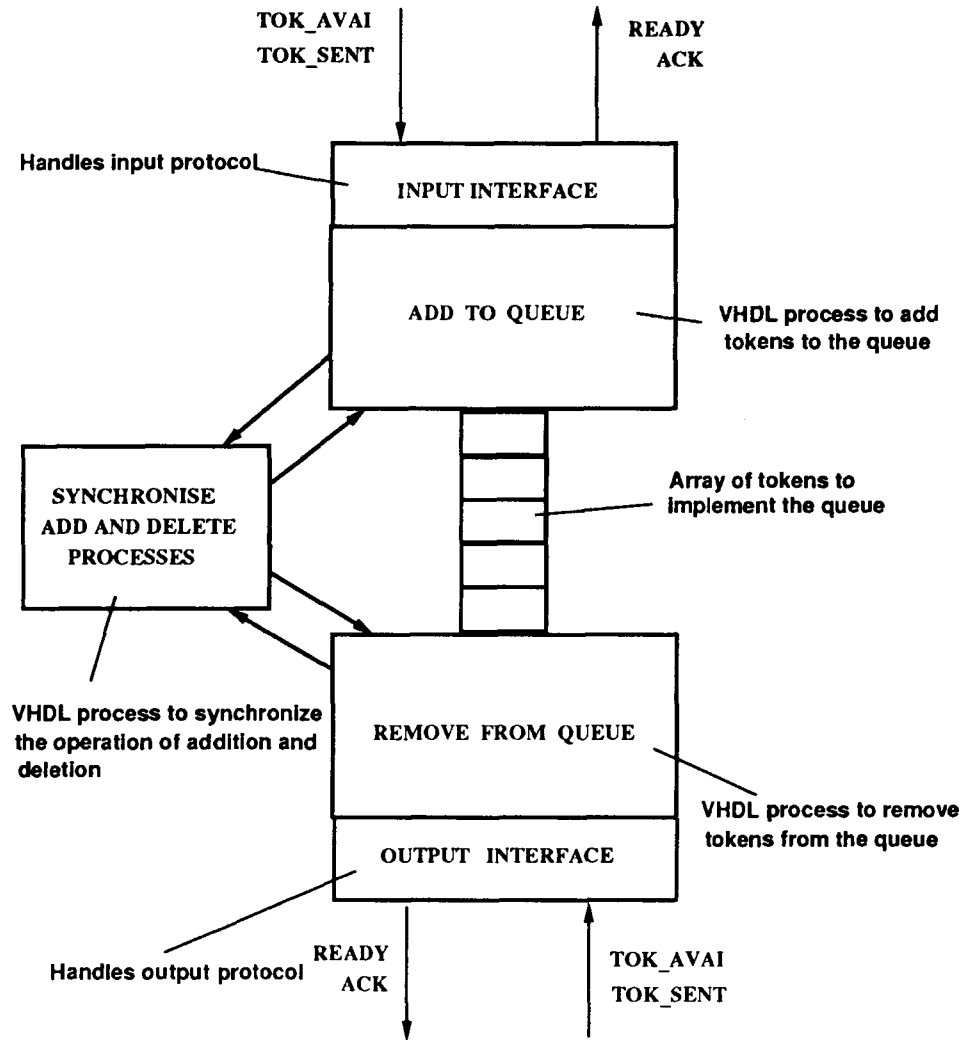


Figure 6: Anatomy of the queue module.

the first point to the second point as specified by the user and the utilization of the set of modules connected between the specified points. The time field of the token is used maintain the state of these measurements. An example of placement of a pair of BETWEEN probes is shown in Figure 7.

Conclusions

The use of VHDL as an intermediate form has been a tradition at the University of Cincinnati to permit rapid development of significant tools for evaluating and simulating special-purpose systems. PERFSIM follows this tradition.

PERFSIM continues to be refined to be more useful and powerful. It has recently been augmented and rewritten and now exists as PERFSIM2 [4]. It is available from the VHDL repository via anonymous ftp from thor.ece.uc.edu.

Acknowledgments

PERFSIM development was motivated by a phase 1 SBIR with MTL Systems, Inc. to study efficient parallel computing systems for Air Force countermeasure effects simulation. Vantage Analysis, Inc., provided the VHDL simulator used to develop PERFSIM.

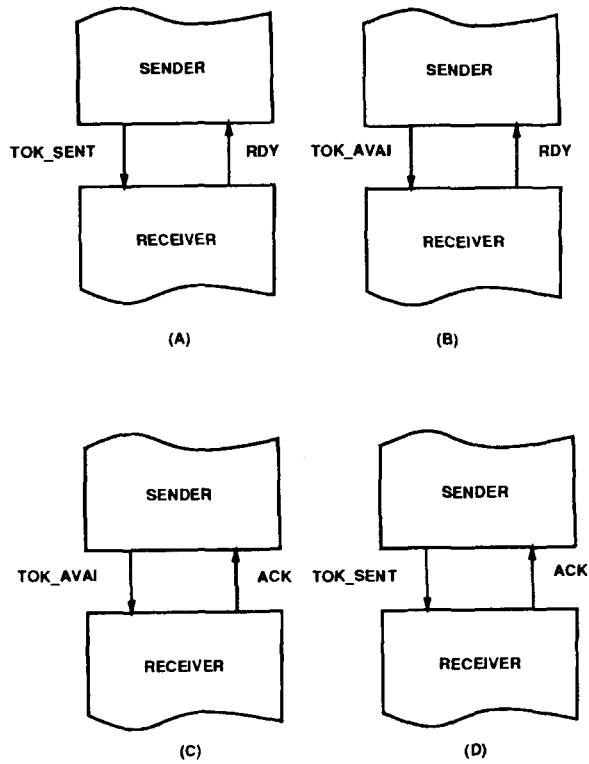


Figure 4: Token passing mechanism.

References

1. Eric D. Cutright, Ramesh Rao, Barry Johnson, James Aylor, "Modeling an ATAMM-Based Multiprocessor System using VHDL," University of Virginia, 1991.
2. Guru Prasad, "An Approach to Performance Analysis of Computer Systems by High Level simulation in VHDL," Master's Thesis, ECE Department, University of Cincinnati, Aug 1993.
3. MTL Systems, Inc., "Multi-Threat Engagement Simulator(MTES)," SBIR 1 Final Report, MTL Doc. MFR-92-003/SDF214-DO1-Revision 001, Dec 1992.
4. Peter Frey, "A Digital System Performance Simulator - PERFSIM2," University of Cincinnati Tech Rep ECE-93/12, Aug 1993.

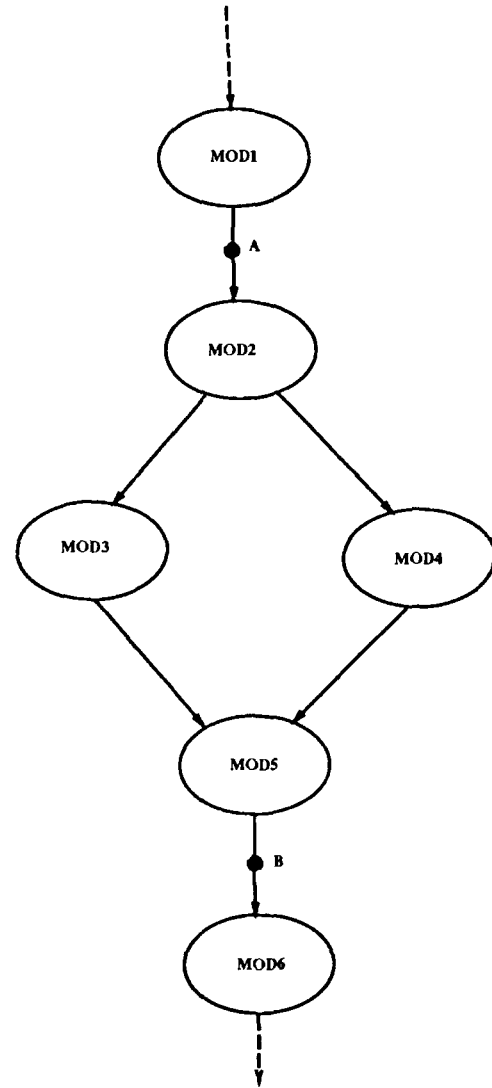


Figure 7: Measurement of time taken by a token to travel from probe point A to probe point B