

Module 59 - Lab : Token-Based Performance Modeling Using VHDL

ATL Performance Modeling Environment Tutorial

Copyright 1995-1999 SCRA

All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute (ATI), and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work belong to other copyright holders and are used with their permission. All information contained, may be duplicated for non-commercial educational use only provided this copyright notice and the copyright acknowledgements herein are included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

The United States Government holds “Unlimited Rights” in all data contained herein under Contract F33615-94-C-1457. Such data may be liberally reproduced and disseminated by the Government, in whole or in part, without restriction except as follows: Certain parts of this work to other copyright holders and are used with their permission; This information contained herein may be duplicated only for non-commercial educational use. Any vehicle, in which part or all of this data is incorporated into, shall carry this notice .

See the [RASSP Disclaimer file](#) for additional RASSP Disclaimer, Warranty and Limitation of Liability Information concerning the material, VHDL code and software developed under the RASSP programs or incorporated in RASSP material.

ATL Performance Modeling Environment Tutorial

1. Overview

- 1.1. This lab will introduce performance modeling using the ATL library elements. The user will simulate a simple 4 processor Mercury MCV6 multicomputer system which has been configured to perform a 2D Fast Fourier Transform (fft). Two different architectural alternatives (as shown in the performance modeling module) will be investigated. The user can then modify the software configuration to determine its affect on performance.

Recall that the first architectural alternative has a single I/O processor that performs both the source and sink functions. The second architectural alternative has two separate processors that perform the source and sink functions in parallel.

Note that in this tutorial, the path to the place where the ATL lab material has been copied and untared is referred to as: *<your_path>*, and the user's home directory is referred to as: *<your_home>*.

2. Getting started

- 2.1. Create a directory for the performance modeling lab material:

```
UNIX>> mkdir perf_modeling
UNIX>> cd perf_modeling
```

- 2.2. Copy the atl_lab files to that directory.

```
UNIX>> cp -r <your_path>/perf_modeling_mod/atl_lab .
```

Note that the space and the, at the end of the command line must not be omitted.

- 2.3. Change to the atl_lab directory and setup for the Mentor Graphics' VHDL environment:

```
UNIX>> cd atl_lab
UNIX>> . bin/mentor_setup
```

- 2.4. A Unix Makefile has been setup to compile all of the required VHDL files. Perform the compilation:

```
UNIX>> make
```

Copyright Ó 1995-1999 SCRA

See first page for copyright notice,
Distribution restrictions and disclaimer

A number of compilation messages should appear. Any errors should be reported to the lab instructors.

3. Examine the source files

3.1. List the contents of the atl_lab directory:

```
UNIX>> ls
```

There are several file and directories listed. The files are the *Makefile* which contains the proper commands to compile the VHDL source code, and the *wave.do* file which contains the simulator commands to generate a wave form display of the token signals (as described below). The contents of the directories are as follows:

- COMPONENTS - contains the VHDL files for the PE, XBAR, and MCV6 (both the alt1 and alt2 architecture) components.
- PACKAGES - contains the token, fifo, and debug VHDL packages used by the above components.
- bin - contains the shell scripts and post-processing executable files.
- programs_alt1 - contains the programs for the architectural alternative 1 components to run the scenario used in the tutorial.
- programs_alt2 - contains the programs for the architectural alternative 2 components to run the scenario used in the tutorial.
- results_alt1 - will contain the ".dat" simulation data files generated by the PE elements in architectural alternative 1 during simulation. This directory should be empty until the initial simulation is run.
- results_alt2 - will contain the ".dat" simulation data files generated by the PE elements during in architectural alternative 2 during simulation. This directory should be empty until the initial simulation is run.
- routes - contains the route files needed by the PE and XBAR elements to route messages to specific processor numbers. These files are identical for both architectural alternatives.
- work - the directory created to hold the compiled VHDL files when the initial "make" command was run.
- docs - contains the postscript version of this document.

- 3.2. It is recommended that the user take a few moments to browse through the files in the COMPONENTS directory and the programs_alt1 and programs_alt2 directories.

```
UNIX>> cd COMPONENTS
UNIX>> more *.vhd1
UNIX>> cd ../programs_alt1
UNIX>> more *.dat
UNIX>> cd ../programs_alt2
UNIX>> more *.dat
UNIX>> cd ..
```

Note that in the COMPONENTS directory there are two mcv6 files, *mcv6_alt1.vhd1* and *mcv6_alt2.vhd1*, one for each architectural alternative. In the programs directories, the program files for the 4 PEs that perform the 2D fft are very similar. The major difference in the programs for the two alternatives is in the programs for the IO processor and the source and sink processors.

4. Perform the simulations

- 4.1. The simulations have been setup to write their results to two separate directories, results_alt1, and results_alt2. The two architectural alternatives will be simulated for 200 ms of real time each. The results will then be examined to determine the throughput of each alternative.

- 4.2. Simulate alternative 1 for 200 ms:

```
UNIX>> qhsim mcv6_alt1
```

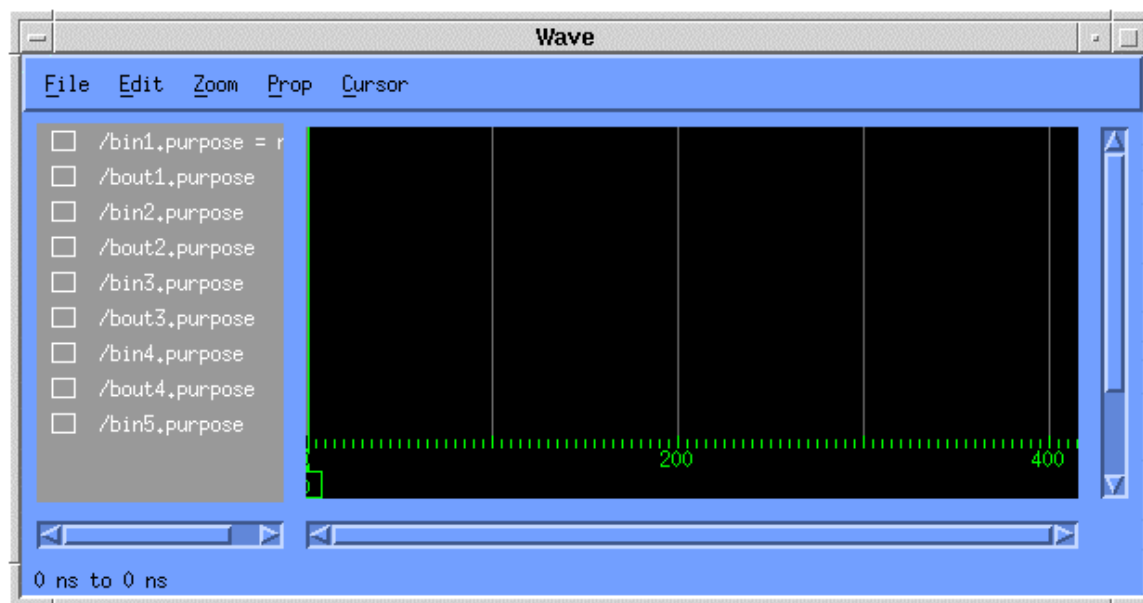
A qhsim.mod window should appear:



4.3. Setup a waveform window to view the token signals that connect the processors to the RACEWay crossbar switch:

QHSIM 1> do wave.do

A wave window which will trace the "purpose" or status fields of the token signals should appear:



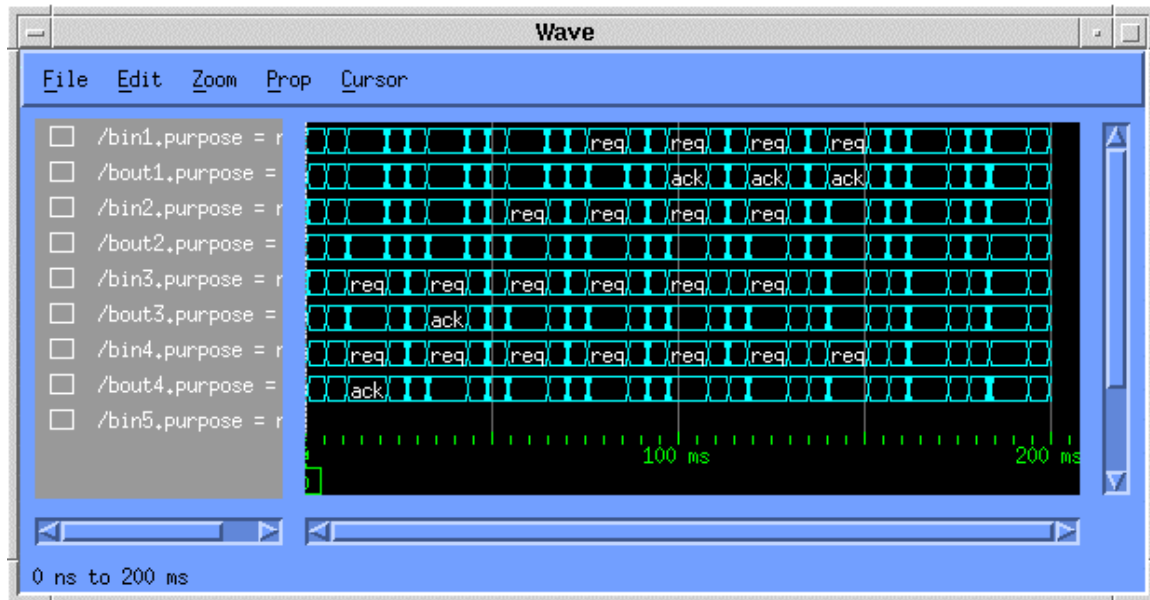
Copyright © 1995-1999 SCRA

See first page for copyright notice,
Distribution restrictions and disclaimer

4.4. Run the simulation for 200 ms:

```
QHSIM 2> run 200 ms
```

In the wave window, use the **Zoom->Full Size** menu item to view the waveform for the signals throughout the simulation time. The wave window should show the purpose field value alternating between "req" and "ack" as tokens are passed through the model:



4.5. Exit the simulation:

```
QHSIM 3> quit
```

Press the **yes** button when the confirm dialog box comes up.

4.6. Simulate the alternate 2 architecture for 200 ms of real time:

```
UNIX>> qhsim mcv6_alt2
```

```
QHSIM 1> run 200 ms
```

```
QHSIM 2> quit
```

Press the **yes** button when the confirm dialog box comes up.

5. Use the post-simulation analysis tools to view the simulation results

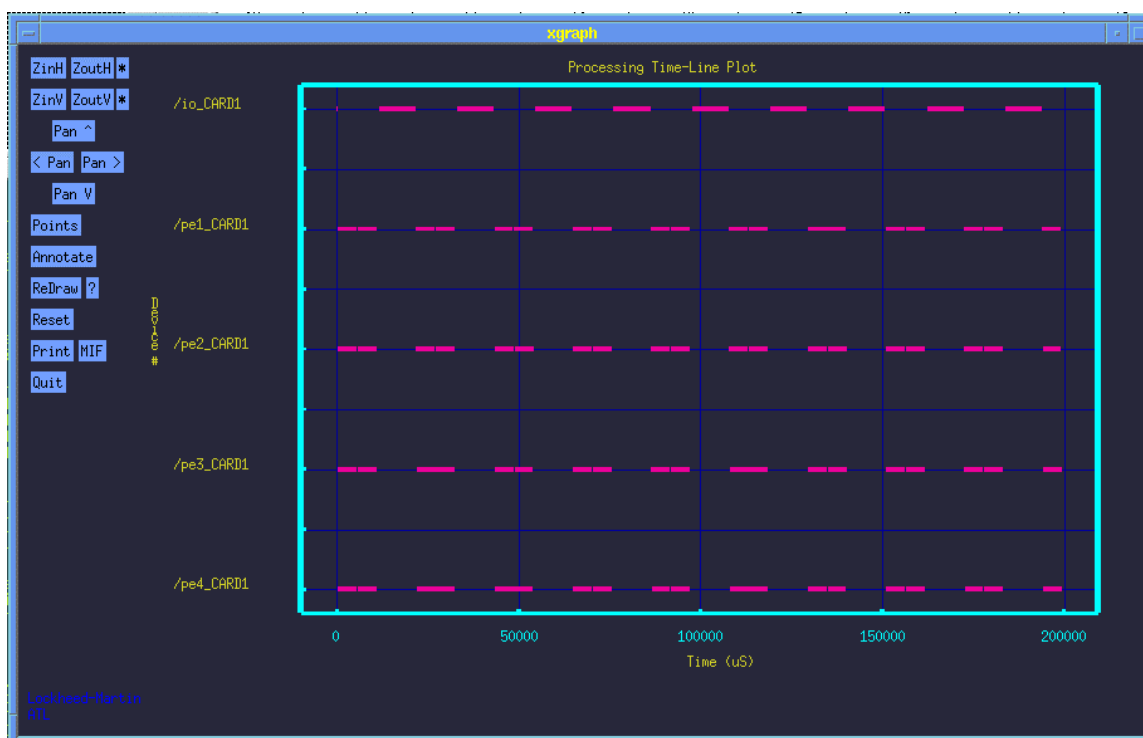
5.1. Shell scripts have been written to process the outputs of the simulations and plot them using the ATL post-processing tools. View the results of architectural alternative 1:

```
UNIX>> bin/plot_alt1
```

Copyright © 1995-1999 SCRA

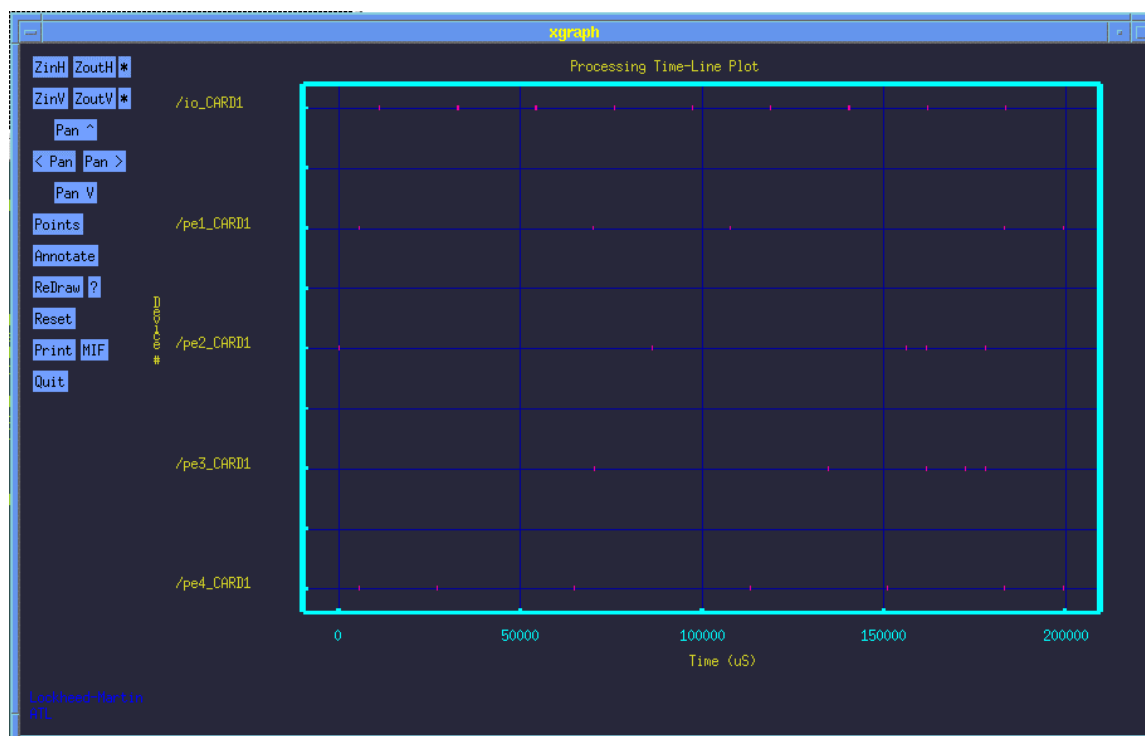
See first page for copyright notice,
Distribution restrictions and disclaimer

The first command in this shell script simply concatenates all of the ".dat" files in the results_alt1 directory into a file called intermed.time in that directory. A number of messages will then scroll past on the screen indicating that the timeline program has been called to generate the *view1.tln* and *view2.tln* timeline files. The *view1.tln* file contains the timeline data for the processor busy times and the *view2.tln* file contains the timeline data for the communication links busy times. Finally, the visualization program **xgraph** is invoked on the *view1.tln* file to show the processor busy time data. The resulting window should look like this:



Note that in this plot, every two computation units for a processing element indicates a completion of a complete 2D fft operation. Therefore, as can be seen from the figure, architecture alternative 1 completes about 9 1/2 iterations of the 2D fft operation in 200 ms.

Press the **Quit** button on the **xgraph** tool when done viewing the results. A second graph of the communications busy times will appear:



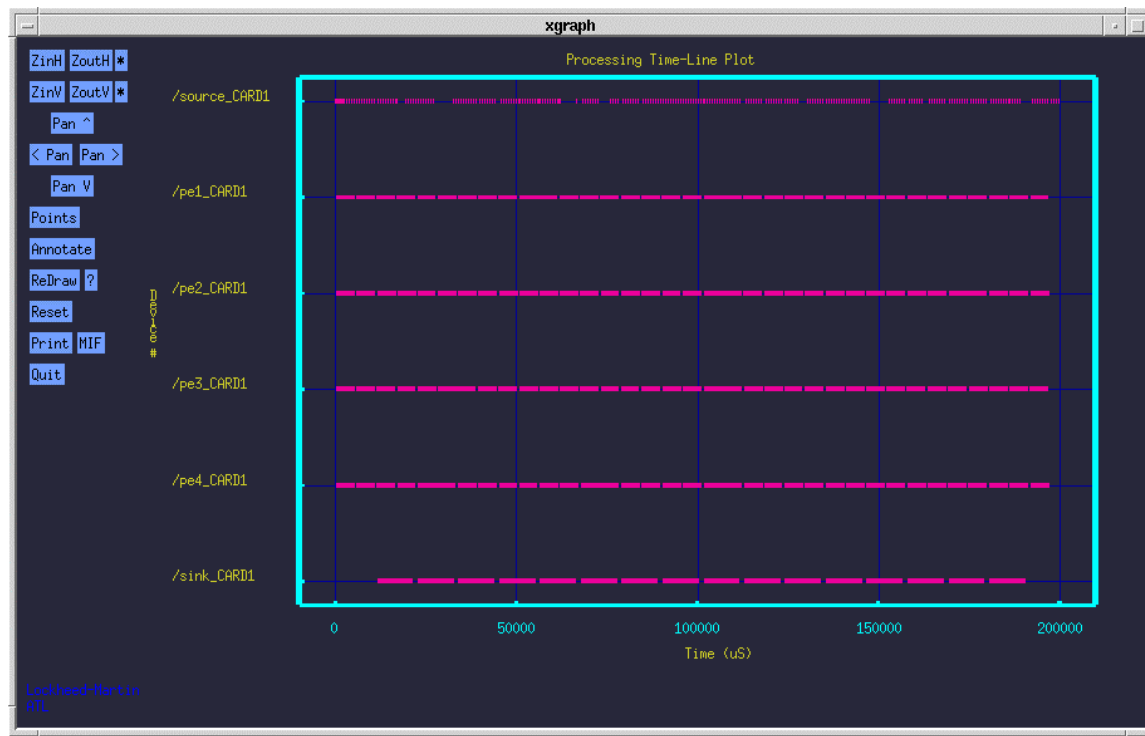
This plot indicates that very little time is spend in actual communication vs. computation. Press the **Quit** button on the xgraph tool when done viewing the results.

```
UNIX>> bin/plot_alt2
```

The post-processing tools will be invoked as before and a plot of the processor busy times will appear:

Copyright © 1995-1999 SCRA

See first page for copyright notice,
Distribution restrictions and disclaimer

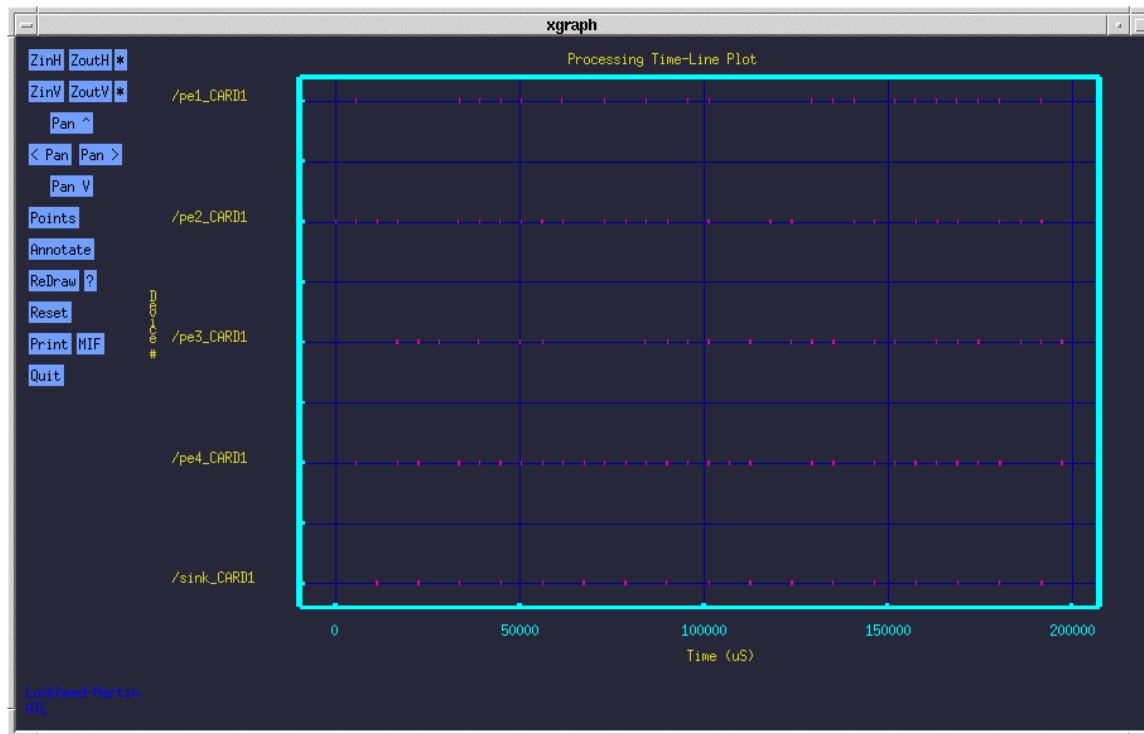


Note that this architectural alternative processes 17 1/2 iterations of the 2D fft operation in the same 200 ms.

The plot of the communication busy times will appear:

Copyright © 1995-1999 SCRA

See first page for copyright notice,
Distribution restrictions and disclaimer



Again, this plot shows that communications time is much less than processing time.

Module 59 - ATL Performance Modeling Environment Exercises

Assignment 1:

Modify the programs for the IO processor in architectural alternative 1 and the sink processor in architectural alternative 2 so that the sink process takes 100 *ms* instead of 10,000 *ms*. Use the post-processing tools to look at the results. Is the performance of alternative 2 significantly better than alternative 1 in this scenario?

Hint - look for the "cecompute 10000 POST_P1_____" command in their program files.

Assignment 2:

Devise a task allocation where instead of parallelizing the individual iterations of the 2D fft operation across 4 processors, the entire operation is performed on a single processor and subsequent iterations of the 2D fft operation are pipelined to the remaining processors. Write programs to perform this task allocation on both architectural alternatives. Simulate and compare the performance of this task allocation on each architecture to each other and to the parallel task allocation. Which architecture and task allocation has the best throughput? What happens to the latency of each 2D fft operation in the pipelined task allocation?