Interactive High-Performance Processor Understanding via the Web

Augustus K. Uht, Sean Langford, and David Morano

***Abstract*— High-performance computer processors have become much more complex in recent years, especially in the research community. We describe a Web-based interactive simulation and graphics tool under development at URI for the new Levo research processor. The tool mimics Levo operation and structure, aiding in broad understanding by researchers, students and engineers.**

***Index Terms*— Microarchitecture, high-performance computing, online simulation, user-defined content.**

I. Introduction

MICROPROCESSOR performance has increased dramatically over the last thirty years. This has unfortunately been accompanied by a similar, if not more rapid, increase in chip transistor count and complexity. Current microprocessors have 10's of millions of transistors. These chips are so complex that one-half or more of the total chip development cost is in *functional verification* (checking its logic). High euro or dollar development cost is just one consequence of the increased complexity: *time-to-market* may become unacceptable. Also, *educating* verification engineers about the chip's operation becomes extremely hard, as does the difficulty of *debugging* the chip (finding and fixing its flaws).

While chip manufacturers can afford to employ hundreds of engineers to design and debug their chips, the microprocessor- or *microarchitecture*-research community cannot. Further, students are limited in their background and the amount of time they can devote to understanding complex designs. Also, it is highly desirable to provide easy access to support tools. Lastly, such tools should be adaptable to completely different processor designs.

In this paper we describe the *LevoVis* Web-based simulation visualization tool. Originally devised specifically for the Levo research processor[6, 7] development effort at the University of Rhode Island (URI) and Northeastern University (NEU), the tool has been modified to be adaptable to any computer architecture or microarchitecture. The remainder of the paper is organized as follows. In Section II other approaches are briefly reviewed. We broadly describe the Levo microarchitecture in Section III. LevoVis itself is described at a high level in Section IV. LevoVis usage is explained in Section V. Examples of LevoVis operation, including a series of screenshots, are given in Section VI. In Section VII we briefly review the future plans for LevoVis. We conclude in Section VIII. Lastly, LevoVis-accessing details are given in Section IX.

This work was supported in part by the U.S. National Science Foundation under Grants Nos. MIP-9708183 and EIA-9729839, the University of Rhode Island Office of the Provost, the Xilinx Corp., and the Mentor Graphics Corp.

Augustus K. Uht is with the University of Rhode Island, Department of Electrical and Computer Engineering, 4 East Alumni Ave., Kingston, RI 02881 USA (telephone: +1-401-874-5431, e-mail: mailto:uht@ele.uri.edu).

Sean Langford was with the University of Rhode Island, and is currently with Experience Inc., One Faneuil Hall Marketplace 3F, Boston, MA 02109 USA (e-mail: langfors@ele.uri.edu).

David Morano is with the Electrical and Computer Engineering Department, Northeastern University, Boston, MA, USA (email: morano@computer.org).

II. Other Approaches

Although there are many systems available for the visualization of microprocessor internals, there are few systems that provide a detailed cycle-by-cycle analysis of execution. Of the few, perhaps the most prolific of these is Rivet[1]. While very powerful, Rivet does not work over the Internet, nor does it provide for easily redefinable views.

III. The Levo Research Processor

Processor performance is composed of two elements, assuming a fixed instruction set: clock frequency f, as in a "2 GHz" Intel Pentium 4 processor, and Instructions Per Cycle *IPC*, which is rarely quoted. Given the above, performance P in terms of instructions per second is:

$$P = IPC * f$$

Note that this is only meaningful when comparing processors with the same instruction set, such as an Intel Pentium III and an AMD Athlon.

Levo is concerned with maximizing *IPC* while still allowing f to be maintained at a high level, thereby improving performance. Levo is still in development, but already exhibits *IPC*'s between 5 and 10 (the norm for a commercial processor is less than 1 *IPC*).

A simplified view of the central Execution Window part of Levo is shown in Fig. 1. Each Active Station, or *AS*, holds one instruction at a time. The Mainline or *M* columns hold regular instructions. The DEE (Disjoint Eager Execution[8]) or *D* columns hold certain speculatively executed instructions. In this version of Levo, every four ASs are grouped together and share one Processing Element or *PE*. (A PE contains all of the arithmetic and Boolean logic necessary to execute the actual operations of a machine instruction.)

Whenever an instruction executes, it broadcasts its result[5] and a novel identifying *time tag* on a bus dedicated to its group. This result is read by later ASs that need the result; once a later result gets a new datum, it executes. Since long busses can slow down a processor (decrease *f* and hence decrease performance), each bus is segmented into electrically separate segments by the Register Forwarding Units or *RFU*s. Each RFU also delays data transmission by one or more cycles, which would decrease performance unacceptably except for a special characteristic of typical code: instruction results are typically used close to where they are produced[2], thus most instructions will not be substantially affected by the RFU delays, keeping performance high.

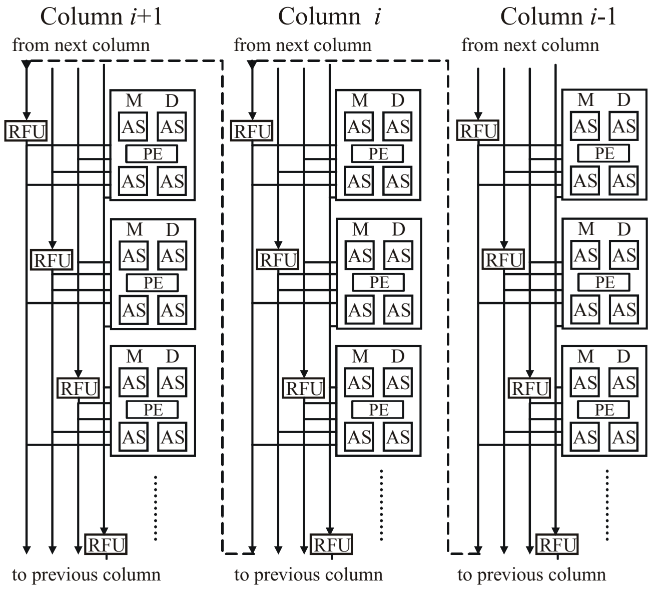


Fig. 1. Levo Execution Window simplified block diagram. Note the regularity of the design; this leads to a machine that is scaleable with respect to number of Processing Elements (adders, etc.). Also, this is a small design for illustrative purposes; a real Levo would be many times larger, leading to an extremely large amount of static and dynamic state to keep track of.

A typical Levo processor would have about 1024 ASs as well as 64 PEs and other supporting hardware. Thus, the amount of internal state or information at any given time is substantial, making understanding and debugging quite challenging: hence our interest in developing LevoVis.

IV. LevoVis Architecture and Overview

In the development of a complicated microprocessor, the flexibility to choose which data should be viewed and the ability to change its view is critical.

The Levo Visualization System is made up of several components. A high-level block diagram of LevoVis is presented in Fig. 2., showing its relationship to the Web (Internet) and supporting elements, such as the LevoSim simulator. The Levo Visualizer renders the state of the Levo processor to the user and takes user input and sends these to the Levo Server. It is automatically downloaded into a user's browser upon the first visit to the Levo Visualizer homepage. The Levo Visualizer Server controls the heavy processes of compilation of user submitted code, simulation of user submitted code and the management of archival storage of previous simulations.

Stored within the visualizer is a graphical representation of the Levo machine. The representation is in the Scalar Vector Graphics (SVG)[10] format, which is an eXtensible Markup Language (XML)-based [9] hierarchical graphics description language. This graphical representation was created using an illustrating program supporting SVG output. The SVG format conveniently provides support for the grouping of graphical elements in a hierarchical way.

The heart of the Visualizer is the tree-traversal algorithm that creates the view from the data. The Levo processor is comprised of several components that fit together hierarchically. The state of these grouped components as revealed through simulation is stored in XML. When initially created the graphical view must implement a similar naming convention[3] for the grouped elements. The tree traversal algorithm begins at the root of both the data and visualization trees and looks for SVG group tags that match node elements within the simulation data. When a match is found, the visual element is rendered; see Fig. 3.

Arbitrary views of Levo can be created using an SVG compliant illustration tool (such as: Adobe Illustrator, CorelDraw, or DIA{Unix}) without touching a single line of LevoVis code. The only requirement is that the same naming convention is followed between the data output and the data view. This also means the actual simulation output may easily be extended, adding new state or even changing simulators completely without requiring any 'retooling' of the Levo Visualizer.

This approach was chosen over using standard graphic library routines because of the flexibility gained. Using (Java's or anyone's) graphical library elements would have made the support of arbitrary views significantly more difficult. It would also require revision of the visualizer client should the Levo simulated state output ever change.

The Levo Visualizer Server feeds the simulation data to the client using RMI[4] (a Java object control and serialization protocol) in chunks of 10 Levo cycles when requested by the client. The simulation files are quite large, holding thousands of cycles worth of Levo state, using several hundred megabytes of disk space. A cycle index mechanism is used to quickly find any requested cycle. The server is also responsible for compiling user-submitted code and activating simulation of all code on the LevoSim simulator.

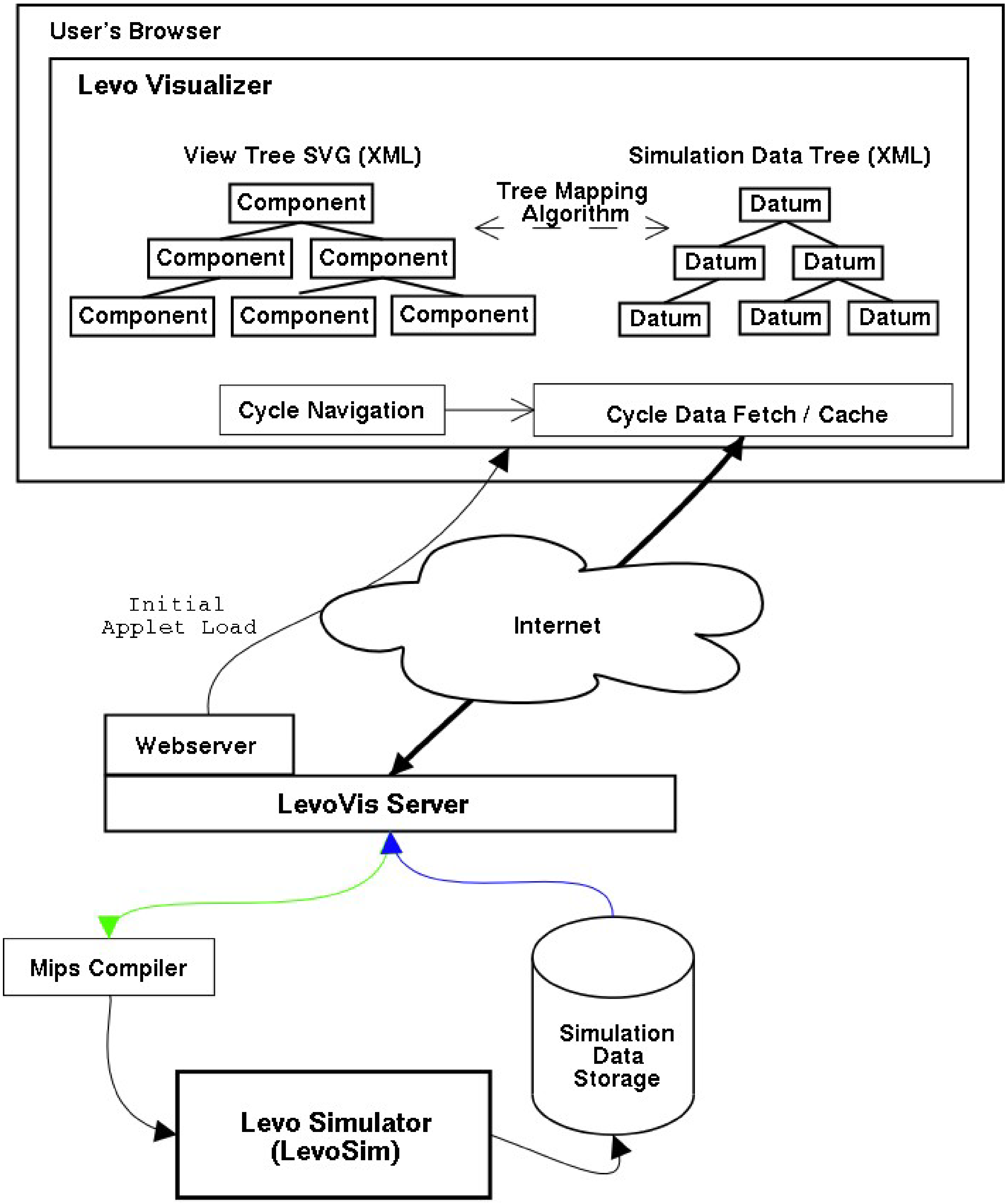


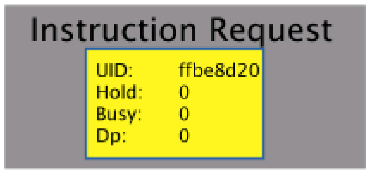
Fig. 2. LevoVis system block diagram. The part above the "Internet" resides on the user's machine (client); the part below resides on the server.

```
Simulation Data XML       SVG Graphic Component XML

<busname>                 <g id="busname" style="stroke:none;">
                              <path style="fill:#919191;" d="M518.404,816.771H361.103v-71.47h157.302v71.47z" />

    <name>                    <g id="name">
                                  <text transform="matrix(1 0 0 1 372.1318 760.8867)">
        Instruction Request           <tspan x="0" y="0" style="font-size:14;">Instruction Request</tspan>
                                  </text>
    </name>                   </g>
    <rfbus>                   <g="rfbus">
                              <path style="fill-rule:evenodd;clip-rule:evenodd;fill:#FAF523;stroke:#2F63AB;stroke-
                                  width:1;stroke-miterlimit:4;" d="M473.741,812.622h-
                                  77.443v-48.121h77.443v48.121z" />
                              <text transform="matrix(1 0 0 1 400.8115 776.5322)">
                                  <tspan x="0" y="0">UID:</tspan>
        <uid>                     <g id="uid">
            ffbe8d20                  <tspan x="36" y="0">ffbe8d20</tspan>
        </uid>                    </g>
                                  <tspan x="0" y="9.5">Hold:</tspan>
        <hold>                    <g id="hold">
            0                         <tspan x="36" y="9.5">0</tspan>
        </hold>                   </g>
                                  <tspan x="0" y="19">Busy:</tspan>
        <busy>                    <g id="busy">
            0                         <tspan x="36" y="19">0</tspan>
        </busy>                   </g>
                                  <tspan x="0" y="28.5">Dp:</tspan>
        <dp>                      <g id="dp">
            0                         <tspan x="36" y="28.5">0</tspan>
        </dp>                     </g>
                              </text>
    </rfbus>                  </g>
</busname>                </g>
```

(a) Illustration of the mapping between simulation data and the SVG graphical representation.



```
Instruction Request
UID:   ffbe8d20
Hold:  0
Busy:  0
Dp:    0
```

(b) The resulting rendered output from (a).

Fig. 3. LevoSim data, the LevoVis SVG graphical representation, and the resulting Levo Visualizer user-viewed output.

V. Using LevoVis

Initially the user has no part of LevoVis on their machine (the *client)*. When the user first contacts the LevoVis Webserver through a standard web browser, the server downloads the initial Java applet client part of LevoVis to the client. The LevoVis Server is then under the control of the client's Levo Visualizer. The view presented in Fig. 4. is a sample view presented to the user.

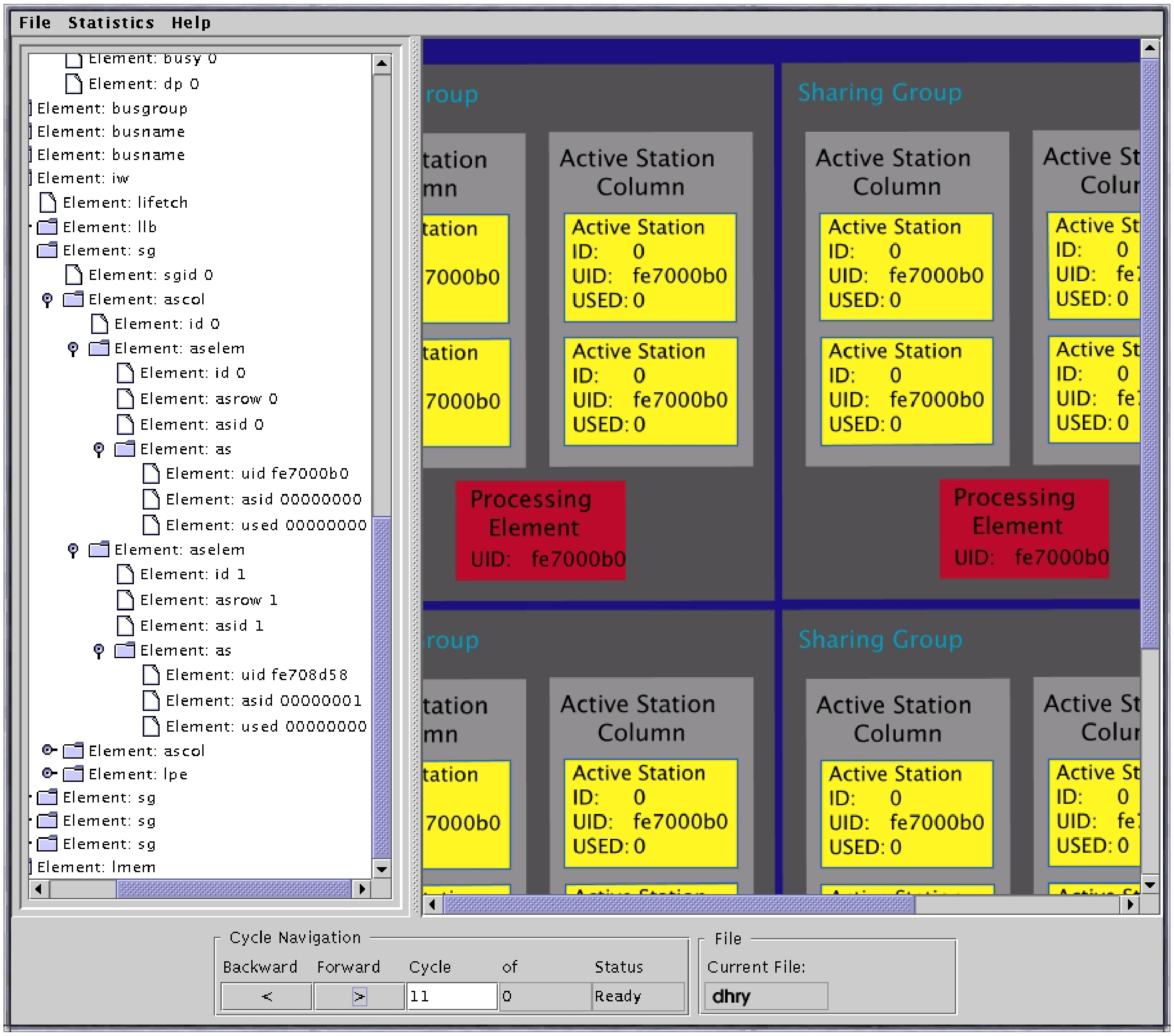
The user is able to go forwards *and backwards* through the code's execution, moving by an arbitrary number of cycles at will. The user is also able to dynamically change what state is displayed. In this way the user is able to understand both the detailed *and* high-level operation of, in this case, the Levo research machine. The graphical representation can be modified to suit any machine or change in Levo's microarchitecture; in such a case, of course, the simulator running on the server would change.

Users can either enter their own C code to be simulated, or select a pre-compiled program to be used. Among the precompiled programs are some of the Spec2000 Integer benchmarks currently used in development of the Levo processor. In the case of a custom program, the code is primarily compiled into MIPS-1 binary or *machine* code by a standard Silicon Graphics compiler and stored on the LevoVis Server. In either code case, the machine code is fed into the LevoSim program. LevoSim is a cycle-accurate simulator running on the LevoVis Server. LevoSim simulates the internal operation of Levo by executing the machine code as if LevoSim contained real hardware. For every cycle of simulation, LevoSim saves the internal Levo state on the server's disk.

VI. LevoVis Examples (Screenshots)

We now present several examples of what a user would see in their Web browser when running LevoVis. Four somewhat different screenshots are shown in Fig. 4. through Fig. 7. (Fig. 4. - Fig. 6. are concept shots; real data is not shown.)

Fig. 4. shows the basic layout of the Levo Visualizer. There are three major components of the window: the state navigation and selection *TREE* pane on the left, the actual state *DISPLAY* pane on the right, and the small *CYCLE* time and code selection pane on the bottom.



```
Element: busy 0
Element: dp 0
Element: busgroup
Element: busname
Element: busname
Element: iw
Element: lifetch
Element: llb
Element: sg
  Element: sgid 0
  Element: ascol
    Element: id 0
    Element: aselem
      Element: id 0
      Element: asrow 0
      Element: asid 0
      Element: as
        Element: uid fe7000b0
        Element: asid 00000000
        Element: used 00000000
    Element: aselem
      Element: id 1
      Element: asrow 1
      Element: asid 1
      Element: as
        Element: uid fe708d58
        Element: asid 00000001
        Element: used 00000000
  Element: ascol
  Element: lpe
Element: sg
Element: sg
Element: sg
Element: lmem
```

Active Station
ID: 0
UID: fe7000b0
USED: 0

Processing Element
UID: fe7000b0

Cycle: 11 of 0 Status: Ready

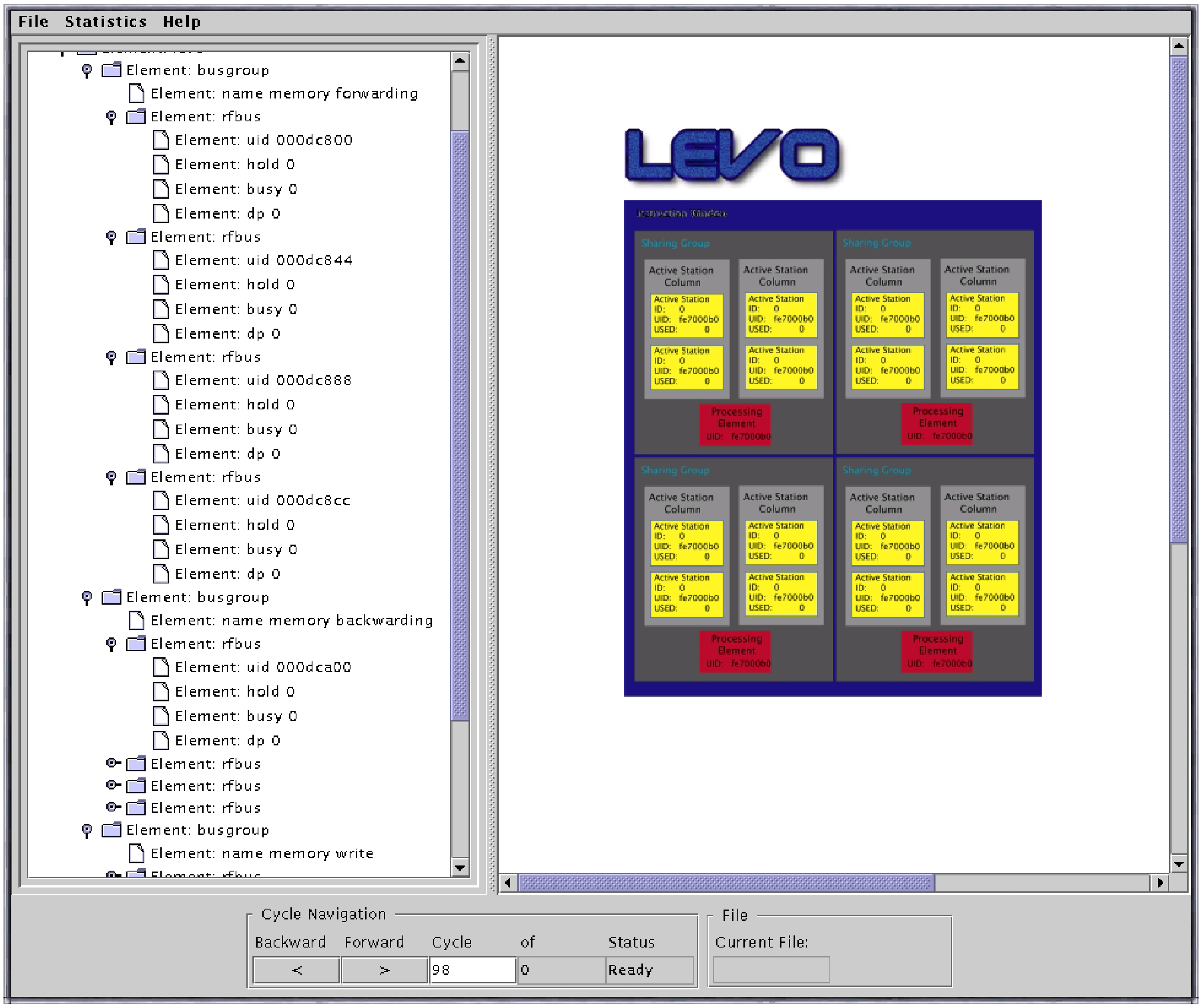
Current File: dhry

Fig. 4. Screenshot of low-level Levo structures and their state as seen on the user's browser. Note the cycle navigation bar at the bottom. Also note the left pane of the screen: it allows the user to select exactly what state to display. (The left pane has a collapsing-tree structure, similar to that used for file systems, ergo Windows Explorer.)

The TREE pane shows the current cycle's state. With this view, the user can quickly find the value of each element. While a particular DISPLAY view may not expose all the data from the simulator at once, the TREE view is guaranteed to allow access to the full state contents for any given cycle. The CYCLE pane navigator component is used to control the view position in the simulation history. The current position can be advanced either by a single cycle forwards or backwards in execution history or an arbitrary time can be selected. This pane also contains a file selector to allow the user to change to a different code simulation.

Fig. 5. shows another view of the window, with a different simulation and with most of the Levo Execution (or Instruction) Window displayed. Fig. 6. is another screenshot, this time also displaying dynamic bus state in the bottom of the DISPLAY pane.

Lastly, Fig. 7. shows the actual state of a Levo simulation at cycle 12 of the execution of the Dhrystone program. (Explaining all of the data is beyond the scope of this paper.)



```
Element: busgroup
  Element: name memory forwarding
  Element: rfbus
    Element: uid 000dc800
    Element: hold 0
    Element: busy 0
    Element: dp 0
  Element: rfbus
    Element: uid 000dc844
    Element: hold 0
    Element: busy 0
    Element: dp 0
  Element: rfbus
    Element: uid 000dc888
    Element: hold 0
    Element: busy 0
    Element: dp 0
  Element: rfbus
    Element: uid 000dc8cc
    Element: hold 0
    Element: busy 0
    Element: dp 0
Element: busgroup
  Element: name memory backwarding
  Element: rfbus
    Element: uid 000dca00
    Element: hold 0
    Element: busy 0
    Element: dp 0
  Element: rfbus
  Element: rfbus
  Element: rfbus
Element: busgroup
  Element: name memory write
```

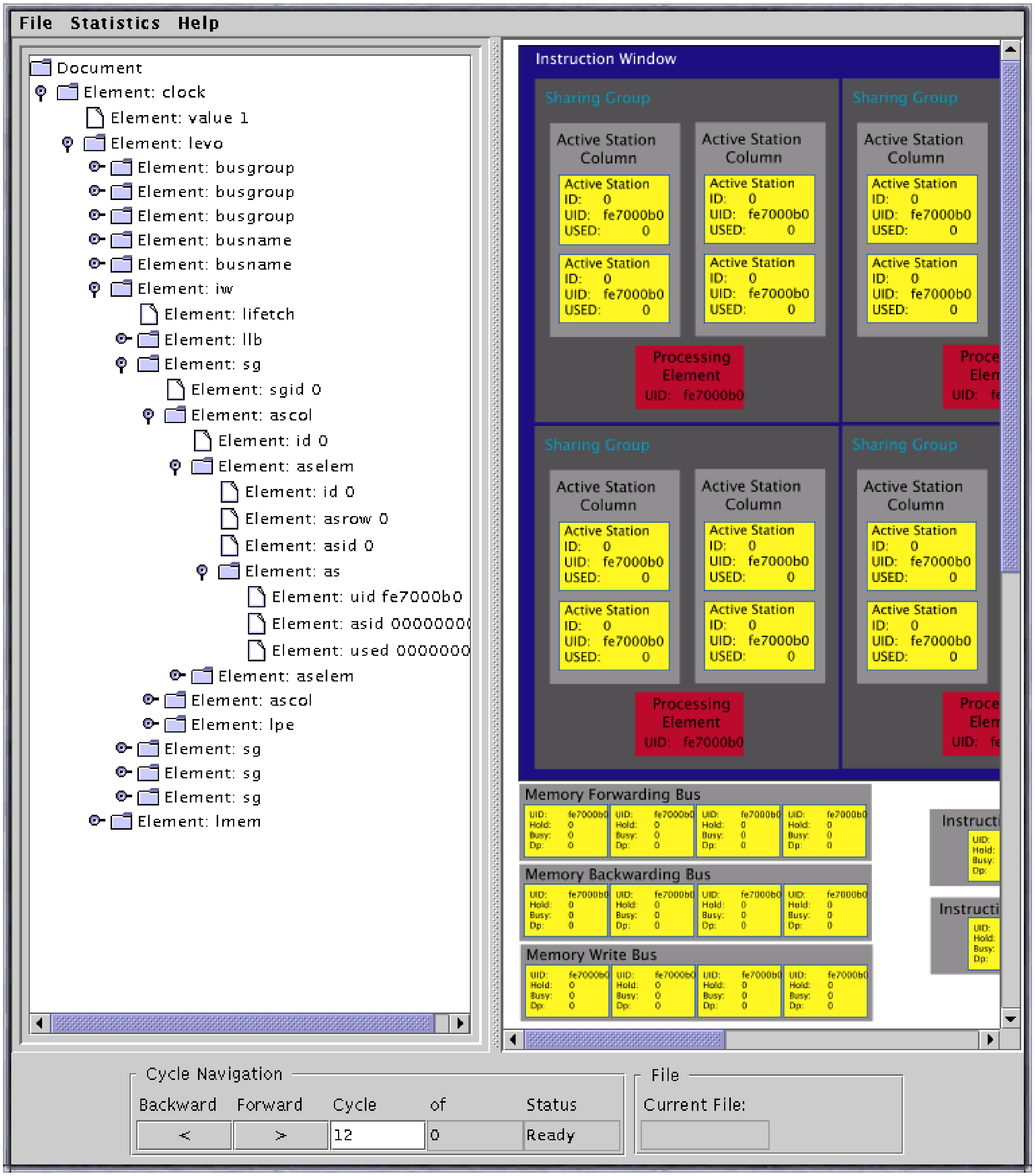
```
Active Station
ID:    0
UID:   fe7000b0
USED:        0

Processing Element
UID:   fe7000b0
```

Cycle: 98 of 0 Status: Ready

File — Current File:

Fig. 5. Levo Visualizer screen shot showing more of a particular configuration of Levo.



```
Document
  Element: clock
      Element: value 1
    Element: levo
      Element: busgroup
      Element: busgroup
      Element: busgroup
      Element: busname
      Element: busname
      Element: iw
          Element: lifetch
        Element: llb
        Element: sg
            Element: sgid 0
          Element: ascol
              Element: id 0
            Element: aselem
                Element: id 0
                Element: asrow 0
                Element: asid 0
              Element: as
                  Element: uid fe7000b0
                  Element: asid 0000000(
                  Element: used 0000000
            Element: aselem
          Element: ascol
          Element: lpe
        Element: sg
        Element: sg
        Element: sg
      Element: lmem
```

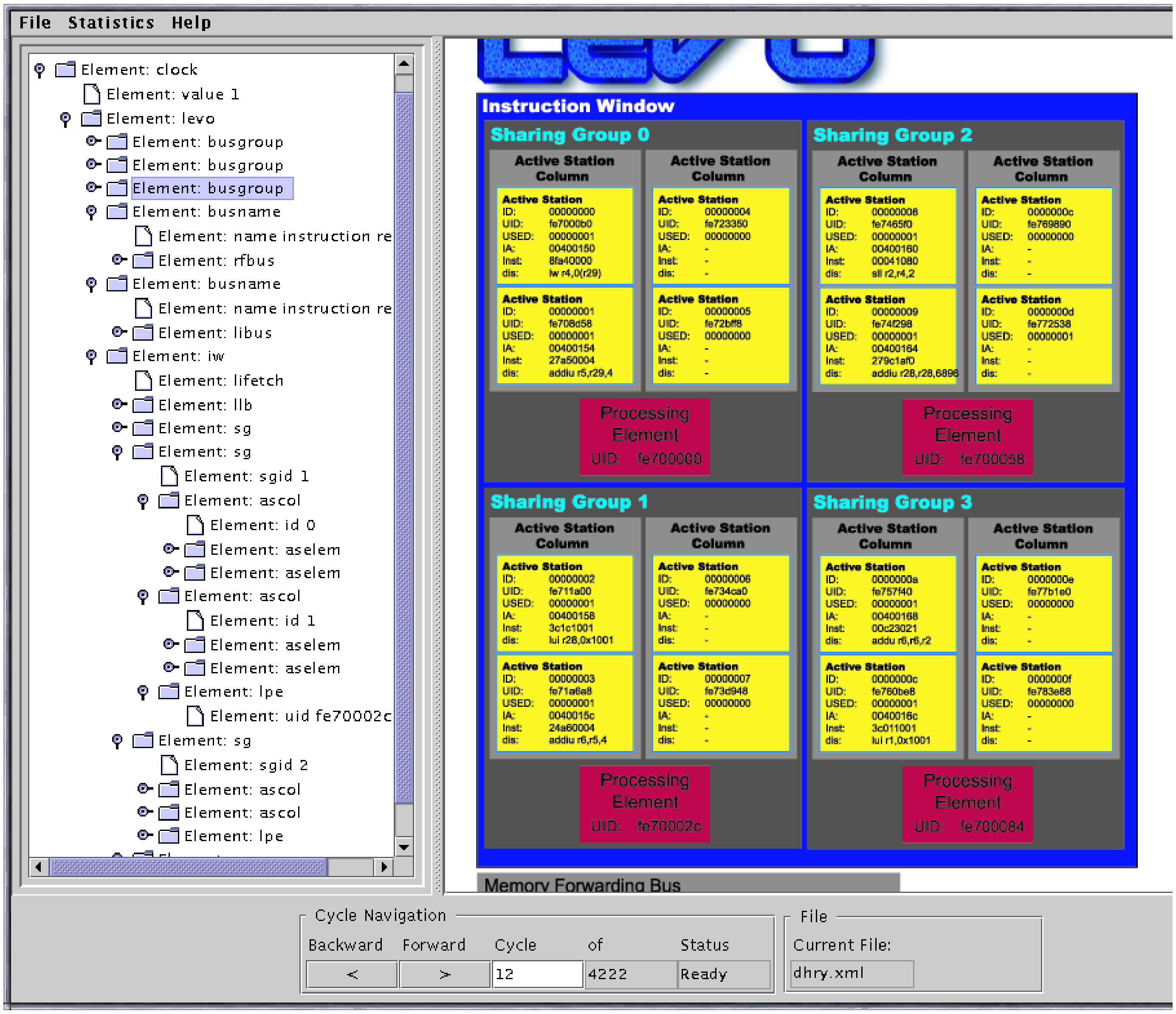
```
Active Station
ID:    0
UID:   fe7000b0
USED:  0
```

```
Processing Element
UID:   fe7000b0
```

```
UID:   fe7000b0
Hold:  0
Busy:  0
Dp:    0
```

Cycle	of	Status
12	0	Ready

Fig. 6. Screenshot showing both Levo static state (that in registers, e.g., ASs) and dynamic state (that on wires or busses, e.g., the Memory Forwarding Bus towards the bottom of the right pane). The "Instruction Window" on the right is the same as the "Execution Window" discussed earlier.



```
Element: clock
    Element: value 1
  Element: levo
    Element: busgroup
    Element: busgroup
    Element: busgroup
    Element: busname
        Element: name instruction re
      Element: rfbus
    Element: busname
        Element: name instruction re
      Element: libus
    Element: iw
        Element: lifetch
      Element: llb
      Element: sg
      Element: sg
          Element: sgid 1
        Element: ascol
            Element: id 0
          Element: aselem
          Element: aselem
        Element: ascol
            Element: id 1
          Element: aselem
          Element: aselem
        Element: lpe
            Element: uid fe70002c
      Element: sg
          Element: sgid 2
        Element: ascol
        Element: ascol
        Element: lpe
```

Sharing Group 0

```
Active Station Column         Active Station Column
Active Station                Active Station
ID:    00000000               ID:    00000004
UID:   fe7000b0               UID:   fe723350
USED:  00000001               USED:  00000000
IA:    00400150               IA:    -
Inst:  8fa40000               Inst:  -
dis:   lw r4,0(r29)           dis:   -
Active Station                Active Station
ID:    00000001               ID:    00000005
UID:   fe708d58               UID:   fe72bff8
USED:  00000001               USED:  00000000
IA:    00400154               IA:    -
Inst:  27a50004               Inst:  -
dis:   addiu r5,r29,4         dis:   -
Processing Element
UID:   fe700000
```

Sharing Group 2

```
Active Station Column         Active Station Column
Active Station                Active Station
ID:    00000008               ID:    0000000c
UID:   fe7465f0               UID:   fe769890
USED:  00000001               USED:  00000000
IA:    00400160               IA:    -
Inst:  00041080               Inst:  -
dis:   sll r2,r4,2            dis:   -
Active Station                Active Station
ID:    00000009               ID:    0000000d
UID:   fe74f298               UID:   fe772538
USED:  00000001               USED:  00000001
IA:    00400164               IA:    -
Inst:  279c1af0               Inst:  -
dis:   addiu r28,r28,6896     dis:   -
Processing Element
UID:   fe700058
```

Sharing Group 1

```
Active Station Column         Active Station Column
Active Station                Active Station
ID:    00000002               ID:    00000006
UID:   fe711a00               UID:   fe734ca0
USED:  00000001               USED:  00000000
IA:    00400158               IA:    -
Inst:  3c1c1001               Inst:  -
dis:   lui r28,0x1001         dis:   -
Active Station                Active Station
ID:    00000003               ID:    00000007
UID:   fe71a6a8               UID:   fe73d948
USED:  00000001               USED:  00000000
IA:    0040015c               IA:    -
Inst:  24a60004               Inst:  -
dis:   addiu r6,r5,4          dis:   -
Processing Element
UID:   fe70002c
```

Sharing Group 3

```
Active Station Column         Active Station Column
Active Station                Active Station
ID:    0000000a               ID:    0000000e
UID:   fe757f40               UID:   fe77b1e0
USED:  00000001               USED:  00000000
IA:    00400168               IA:    -
Inst:  00c23021               Inst:  -
dis:   addu r6,r6,r2          dis:   -
Active Station                Active Station
ID:    0000000c               ID:    0000000f
UID:   fe760be8               UID:   fe783e88
USED:  00000001               USED:  00000000
IA:    0040016c               IA:    -
Inst:  3c011001               Inst:  -
dis:   lui r1,0x1001          dis:   -
Processing Element
UID:   fe700084
```

Cycle Navigation

Backward	Forward	Cycle	of	Status
<	>	12	4222	Ready

File

Current File: dhry.xml

Fig. 7. Last screenshot, showing actual static state of the Levo ASs and PEs for cycle 12 of the execution of the industry-standard Dhrystone performance test program.

VII. ONGOING ENHANCEMENTS

Clearly, LevoVis opens a wealth of possibilities for enhancing the visualization of processors. Some of our current work and ongoing enhancements includes:

- finish on the fly code compilation / submission
- add inline stream compression of the data between the client and server to decrease network bandwidth consumption
- add ability to select different views on the fly
- allow the use to pop out child windows, each with their own cycle navigator, so different windows can see different cycles for state comparison between cycles
- add a delta indicator (some indication of elements who's state has changed between cycles)
- add statistics gathering.

VIII. SUMMARY

LevoVis provides a wealth of flexibility and state visualization capabilities for complex processors. These allow ready understanding, analysis, and debugging of processors by researchers, students, and engineers working concurrently and remotely from the LevoVis host. That is, world-wide access to current and future processor internal operation is possible.

Further, while our discussion has focused on one machine (Levo), we must emphasize that LevoVis is adaptable to any style of processor; indeed, even to *any* type of synchronous digital system.

IX. LEVOVIS ACCESS

The LevoVis host resides at the Dept. of Electrical and Computer Engineering at the University of Rhode Island,

and may be accessed indirectly or directly, resp., through the URLs:

http://www.levo.org

or:

http://ovel.ele.uri.edu:8080

(Yes, that's Levo backwards, for historical reasons.) It is also accessible indirectly via the authors' web sites, including:

http://www.ele.uri.edu/~uht

ACKNOWLEDGMENT

We are indebted to Laurette Bradley, who first suggested the creation of a Web-based visualizer for Levo.

We are also profoundly grateful to the whole Levo team for their hard work and support.

REFERENCES

[1] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum, and P. Hanrahan, "Rivet: A flexible environment for computer systems visualization," *Computer Graphics - US*, vol. 34, no. 1, pp. 68-73, February 2000.

[2] M. Franklin and G. S. Sohi, "Register Traffic Analysis for Streamlining Inter-Operation Communication in Fine-Grain Parallel Processors," in *Proceedings of the Twenty-Fifth International Symposium on Microarchitecture (MICRO-25)*: IEEE and ACM, December 1992, pp. 236-245.

[3] D. Morano, "Levo State Trace Output Description," unpublished report, 2001, URL: via: http://www.levo.org, to appear.

[4] Sun Microsystems, "Java Programming Language," URL: http://java.sun.com, accessed: November 2001.

[5] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 25-33, January 1967.

[6] A. K. Uht, A. Khalafi, D. Morano, T. Wenisch, M. d. Alba, and D. Kaeli, "Levo: IPC in the 10's via Resource Flow Computing," *in Work-In-Progress session, PACT-2001; appears in a special issue of IEEE TCCA News*, December 2001.

[7] A. K. Uht, D. Morano, A. Khalafi, M. d. Alba, T. Wenisch, M. Ashouei, and D. Kaeli, "IPC in the 10's via Resource Flow Computing with Levo," Department of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI Technical Report 092001-001, September 18, 2001, Available via: http://www.ele.uri.edu/~uht.

[8] A. K. Uht and V. Sindagi, "Disjoint Eager Execution: An Optimal Form of Speculative Execution," in *Proceedings of the 28th International Symposium on Microarchitecture (MICRO-28)*. Ann Arbor, MI, November/December 1995, pp. 313-325.

[9] W3C, "The World Wide Web Consortium (XML) Homepage," URL: http://www.w3.org, accessed: November 2001.

[10] W3C, "The World Wide Web Consortium SVG Homepage," URL: http://www.w3.org/Graphics/SVG/Overview.html, accessed: November 2001.