The Liberty Simulation Environment's Visualizer Manual

The Liberty Research Group

The Liberty Simulation Environment's Visualizer Manual

by The Liberty Research Group

Version 1.0 Edition

Table of Contents

Preface ... i

Typographical conventions used in this book ... i

1. Static Visualization of LSE Configurations ... 1

Basic Functionality ... 1

Starting the Visualizer ... 1

The Visualizer Main Window ... 1

The Visualizer Editor Window ... 2

The Visualizer Schematic View Window ... 6

Customizing the Schematic View ... 9

Customization Primitives ... 10

Properties ... 10

Customizing the Visual Representation of Canvas Components ... 11

Customizing the Visual Representation of Instances ... 11

2. Dynamic Visualization of LSE Configurations ... 12

Basic Functionality ... 12

List of Figures

1-1. The Visualizer Main Window....2
1-2. Visualizer Editor Window....3
1-3. Build Results Dialog....3
1-4. Compilation Dialog....4
1-5. Simulator Build Results Dialog....5
1-6. Execution Dialog....5
1-7. Execution Dialog....5
1-8. Visualizer Schematic Window....6
1-9. Visualizer Schematic Window - Component Pop-Up Menu....8
1-10. Property Editor Dialog....8
1-11. Instance Parameters Dialog....9
2-1. Execution Animation in the Schematic View....13
2-2. Execution Results....13

List of Examples

1-1. Sample Properties....10
1-2. Sample Properties....10
2-1. SchematicFigure Interface Function....12
2-2. DefaultInstanceFigure Commands....12
2-3. Simulator Instrumentation....12
2-4. 13
2-5. 13

Preface

This book describes how to use the LSE Visualizer. The first chapter is dedicated to the static visualization of configurations, more specifically, it is focussed primarily on how to start the visualizer, modify the appearence of modules, instances and connections in a configuration and finally, how and where visualization-specific information is stored. The second chapter is focussed on how the visualizer may be used in conjuction with the simulator in order to visualize animated simulation data.

Typographical conventions used in this book

The following typefaces are used in this book:

- Normal text
- *Emphasized text*
- `The name of a program variable`
- `The name of a constant`
- **The name of an LSE module**
- **The name of a package**
- *The name of an domain class*
- **The name of an domain implementation**
- **The name of an attribute in a domain implementation description file**
- **The name of an emulator**
- *The name of an emulator capability*
- *The name of a module parameter*
- **The name of a module port**
- `Literal text`
- *`Text the user replaces`*
- `The name of a file`
- The name of an environment variable
- *The first occurrence of a term*

Chapter 1. Static Visualization of LSE Configurations

The LSE Visualizer is a tool for visualizing the block structure of an LSS configuration. After the visualizer renders the block diagram, it allows users to layout components, modify their visual representation and store this data for later use.

The purpose of this chapter is to familiarize users with the LSE visualizer, it will demonstrate each of the folloowing:

- How to run the visualizer
- How to modify the visual representation of modules, instances and connections
- How to extend visualization capabilities

Basic Functionality

Starting the Visualizer

The visualizer is started from the command line (provided that ${LIBERTY}/bin is in your PATH environment variable) by issuing the following command:

```
visualizer [options] [lssfile_1, lssfile_2, ...]
```

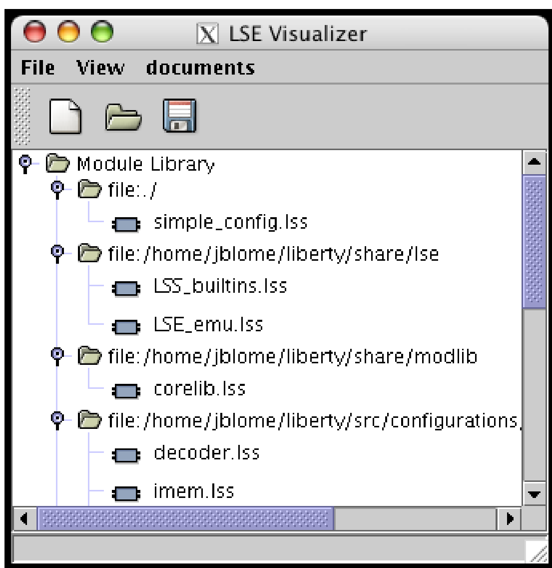
Note that a list of the available options for the visualizer can be viewed by typing the command:

```
visualizer --help
```

Upon issuing the `visualizer` command, the user will be presented with one or more windows. The first window is the visualizer main window, which is shown below in Figure 1-1. Then for each LSS file specified on the command line, a source editor window, as shown in Figure 1-2 will be opened.

The Visualizer Main Window

Figure 1-1. The Visualizer Main Window



The window shown in Figure 1-1 is the main window of the visualizer application. From the main window the user has the ability to open files, create new files, and save the currently focussed file. It is also used to manage open documents and show or hide the different views available to them.

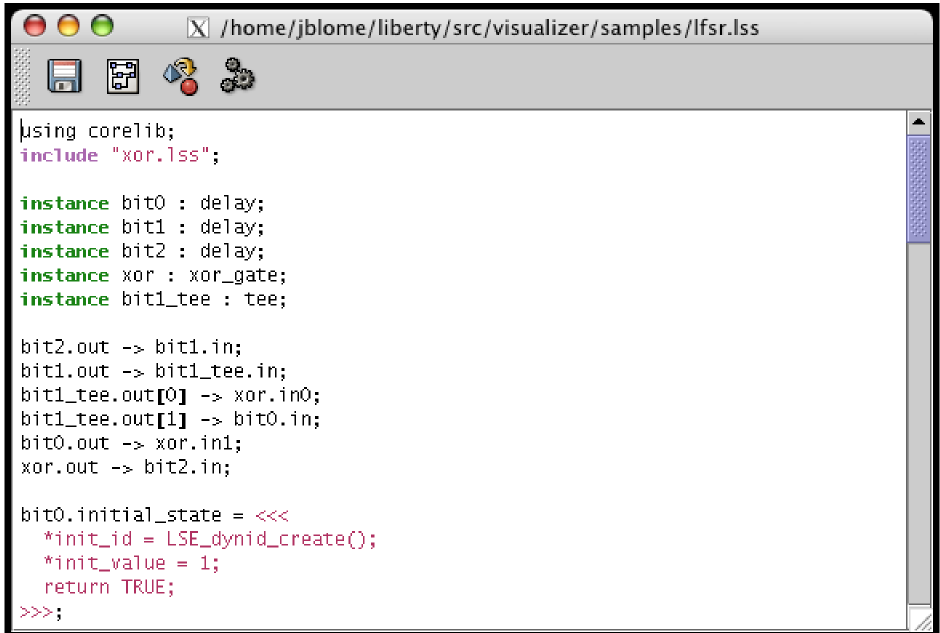
The tree widget contained in this window displays the contents of the user's module library, as specified by the environment variable LIBERTY_SIM_USER_PATH. It is important to note here that the library visible to the visualizer can be augmented by specifying one or both of the following command line options: `--mpathbeg=`*`path`* or `--mpathend=`*`path`*. It is necessary that the library contain the correct directories for building the configuration that has been opened, or the visualizer will not be able to build a schematic representation of the configuration. The user can view files in the module library by simply right clicking on an lss file in the tree, and selecting the option `open file` from the popup menu. The following list of figures details the functionality of the buttons on the Main Window's toolbar.

- This button opens a new file for editing in a source editor window.
- This button will pop up a file chooser dialog from which the user can select a file to open in the visualizer. Upon selecting a file, a new source editor window with the file's contents will be opened.
- This button will save the document that the currently focussed window is associated with.

The Visualizer Editor Window

> **Open Issue**
>
> - The syntax highlighting functionality is mildly outdated in terms of the current LSS version's syntax

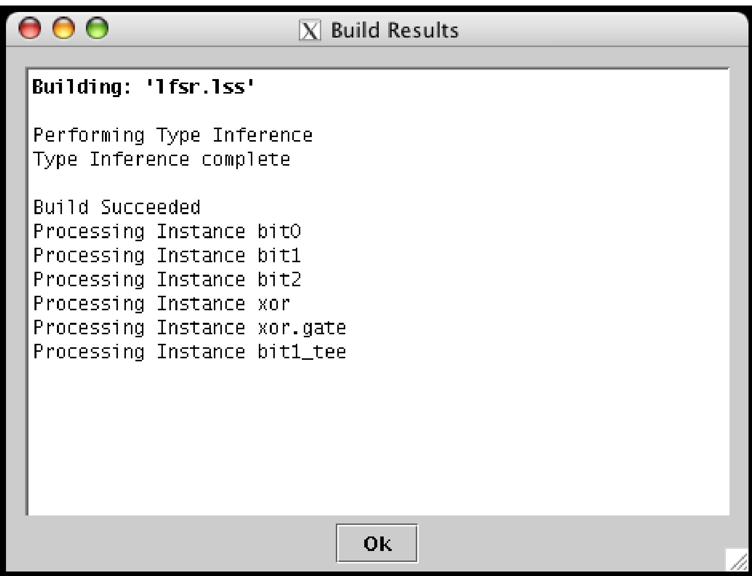
Figure 1-2. Visualizer Editor Window



The window shown in Figure 1-2 is the LSE Visualizer's editor window, which is used to view the source code of LSS files. It provides simple syntax highlighting relevant to the LSS language and allows the user to save file modifications. Here we will list the functionality of each button on this window's toolbar.

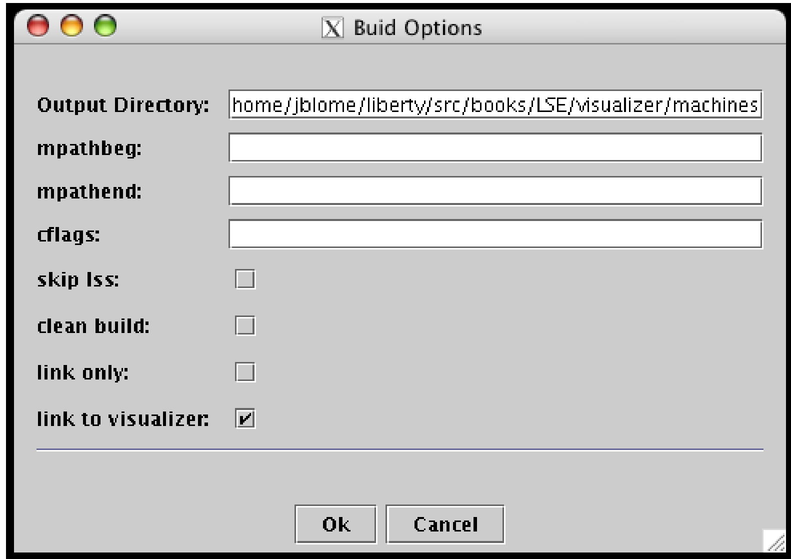
- This button will cause any modifications to the LSS file made in the editor window to be stored back to the file.
- This button will cause the visualizer to compile the LSS file and build a block representation of its structure. The compilation results are displayed in the dialog box shown in Figure 1-3 below.
- This button will pop up a dialog requesting parameters in order to build and link an executable simulator from this document. The dialog requesting parameters is shown below in Figure 1-4 and the build results are displayed in Figure 1-5 below.
- This button will bring up the dialog shown in Figure 1-6 in order to collect the parameters necessary to execute a simulator binary.

Figure 1-3. Build Results Dialog



The above dialog in Figure 1-3 is displaying the results of the file `lfsr.lss`. The text box will show all output of the LSS compilation process as well as the final result of the build process.

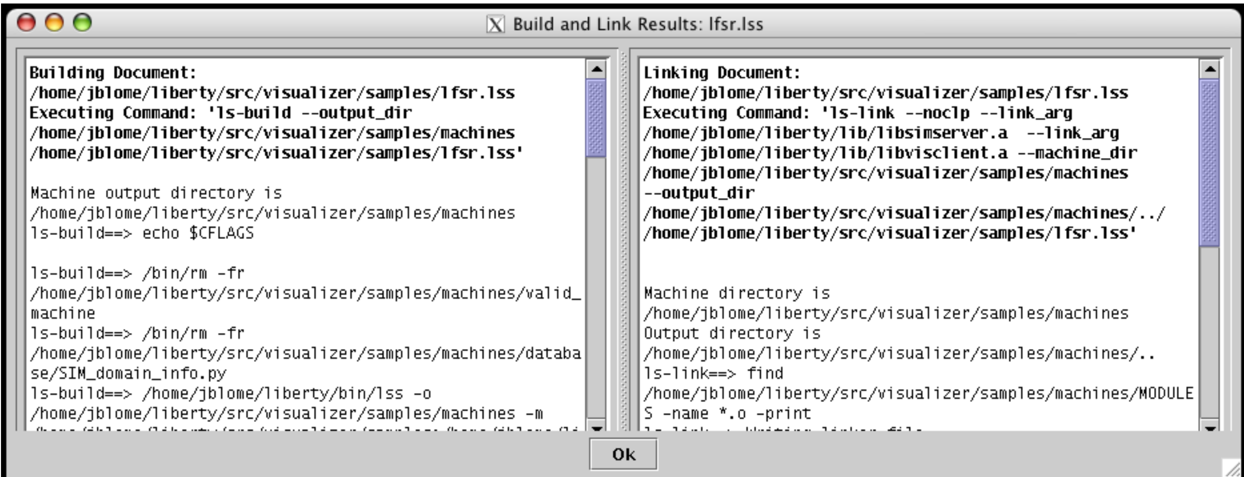
Figure 1-4. Compilation Dialog



The dialog shown in Figure 1-4 is used to gather all of the parameters necessary to build an executable simulator from an LSS file. Users may specify the "Output Directory" where they would like the final simulator executable to be located, the `mpathbeg` and `mpathend` parameters as mentioned in the Section called *Basic Functionality* and any `cflags` that they would like to pass to the compilation process. Also, the user can specify whether the compilation process should skip the LSS compilation phase, perform a clean build, only perform linking operations and whether or not the built simulator should be linked to

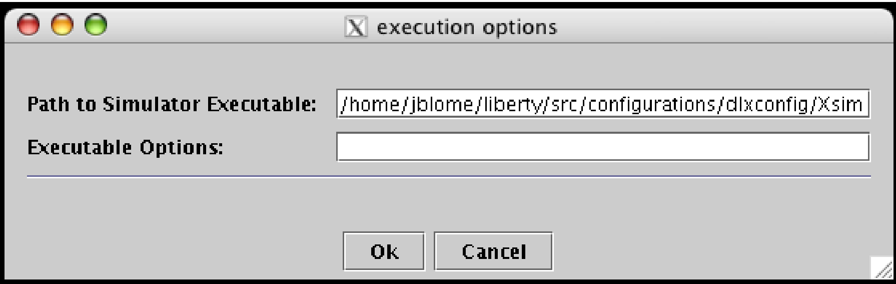
the visualizer's command line processor (CLP). Note that in order to make use of the visualizer's execution animation facilities as described in Chapter 2, the "link to visualizer" option must be selected.

Figure 1-5. Simulator Build Results Dialog



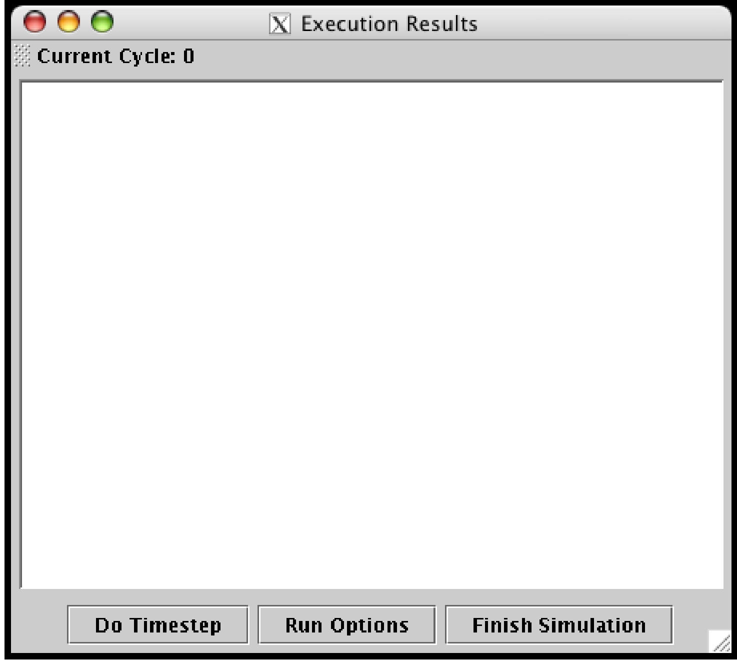
The dialog shown above in Figure 1-5 is used to display the results of clicking the "ok" button on the dialog from Figure 1-4. The two text widgets in this dialog are used to display the build and link results respectively. More specifically, the left widget will display the results of running the ls-build script as shown in bold at the begining of the output. The right widget will display the results of the ls-link script. Both widgets will show output from stdout in black text and output from stderr in red text.

Figure 1-6. Execution Dialog



The execution options dialog in Figure 1-6 is used to gather any parameters necessary to execute a simulator binary. The results of clicking the "ok" button are shown in the Figure 1-7 below.

Figure 1-7. Execution Dialog

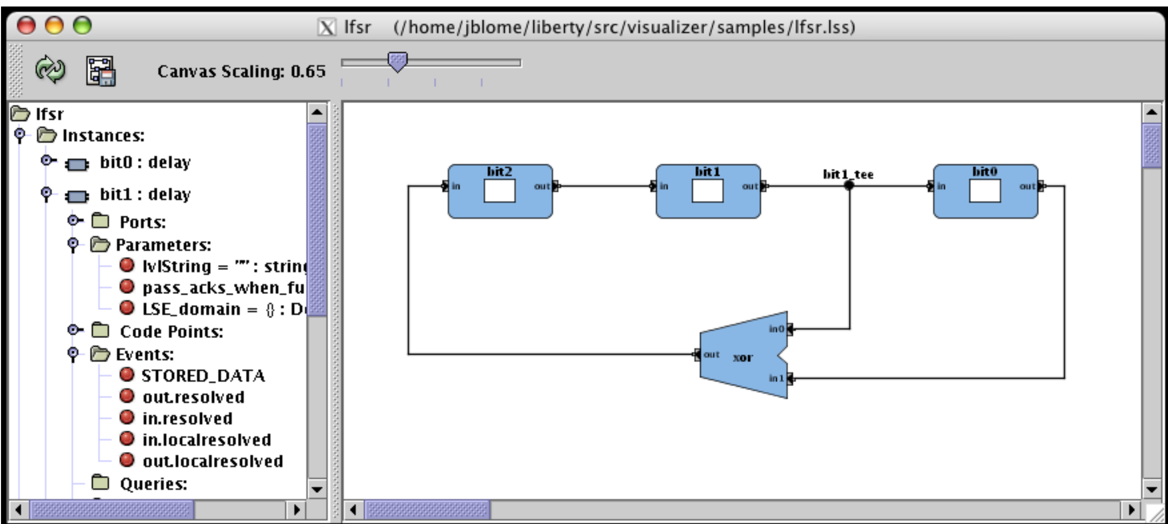


The dialog show above in Figure 1-7 is used to show any output caused by running a simulator binary. It also used to control the execution of the simulator binary. The leftmost button on the bottom of the dialog, labeled "Do Timestep" will cause the simulator to execute one simulation cycle. All buttons on this dialog will be disabled until the simulator finishes execution of the simulation cycle. Also, any output from the simulator will be displayed in the text widget in this dialog. The button labeled "Run Options" will present the user with a number of options for simulation execution. The last button, labeled "Finish Simulation" will finalize the simulation, return the exit value from the binary simulator and kill the simulation server.

The Visualizer Schematic View Window

The LSE Visualizer's schematic view window, shown below in Figure 1-8, is used to display a block diagram representing the structure of an LSS configuration. In this view, the user has the ability to lay out components, customize how each component looks in the diagram, and access any parameterization information about the component.

Figure 1-8. Visualizer Schematic Window



As shown in Figure 1-8, the schematic view is composed of two widgets, a canvas upon which the block diagram is drawn and a tree widget which is used to convey all parameterization information about components in the configuration. The following list describes the functionality of the buttons located in the schematic view's toolbar.

- This button is used to refresh the schematic view. If the source file has changed or, the property file discussed below in the Section called *Customization Primitives* is modified, pressing this button will cause the visualizer to rebuild the LSS document and update this schematic view appropriately.
- This button is used to store the layout and customized rendering options for this configuration so that they will be reloaded the next time this document is opened.
- Canvas Scaling: 0.65 This slider widget is used to scale the block diagram rendered on the canvas.

Now, it is important to note that every element on the canvas has associated with it a popup menu, as does every element in the tree widget. Right clicking on either of these view components will present the user with a pop-up menu similar to the one show below in Figure 1-9 below. Now, each pop-up menu is specific to the component that has been clicked, here, an `instance` has been right clicked, and the user is presented with a menu with five items. The first item "View Visual Properties" will present the user with a property editor dialog similar to the one shown if Figure 1-10. This dialog allows the user to customize how each canvas element is rendered. The next menu item, "View Hierarcy" will only appear on menus associated with hierarchical instances. Clicking on this menu item will open a new schematic view, which shows the internal components of the given instance. The options "View Module Code" and "View Module Source File" will present the user with source editor windows, displaying either only the pertinent code where the module is defined or the entire source file respectively. The final menu item "View Instance Data" will pop up the dialog shown in Figure 1-11 which lists all of the parameterization information about the instance. Note, that all elements on the canvas and in the tree view will have a

similar menu option in their popup menu, and that double clicking on any element, either on the canvas, or on the tree widget, will bring up a similar dialog, listing all data about the given element, be it and instance, port, connection, parameter, code point, etc.

Figure 1-9. Visualizer Schematic Window - Component Pop-Up Menu

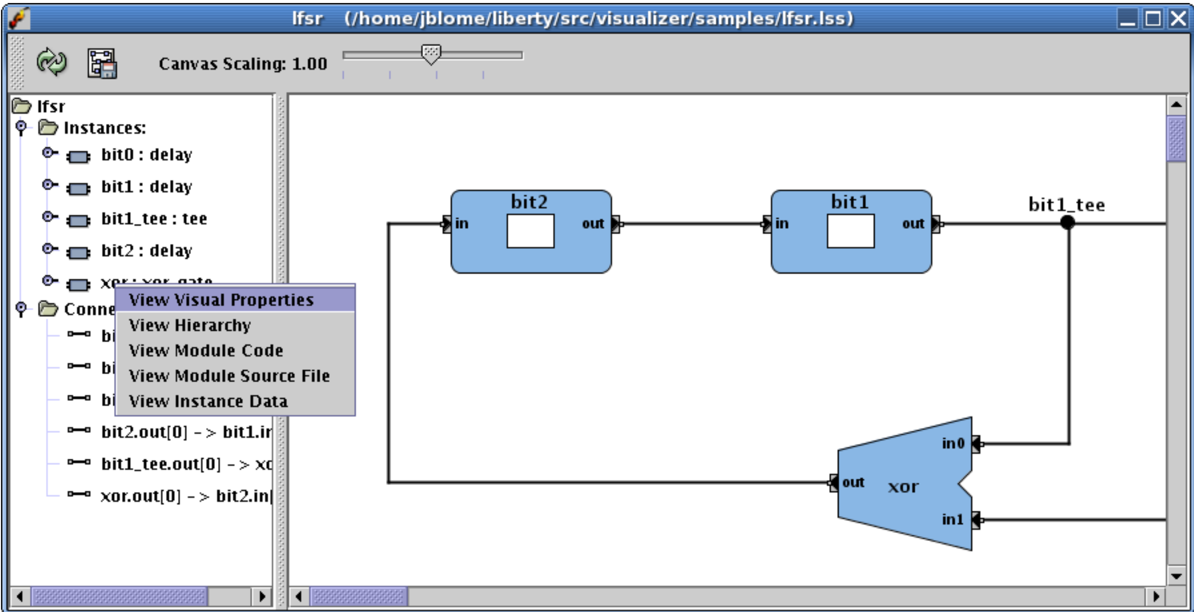


Figure 1-10. Property Editor Dialog

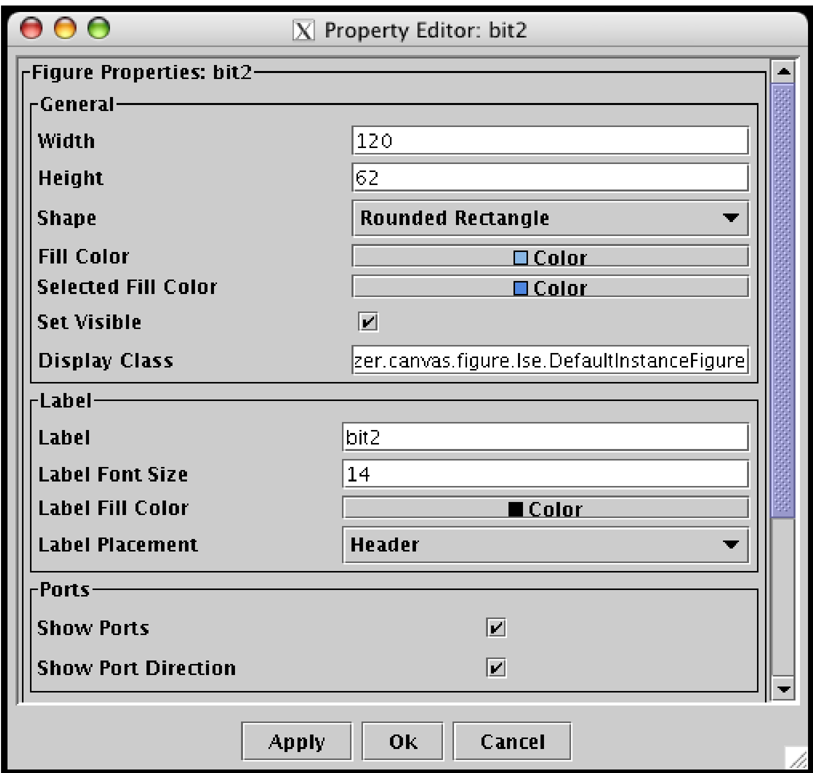
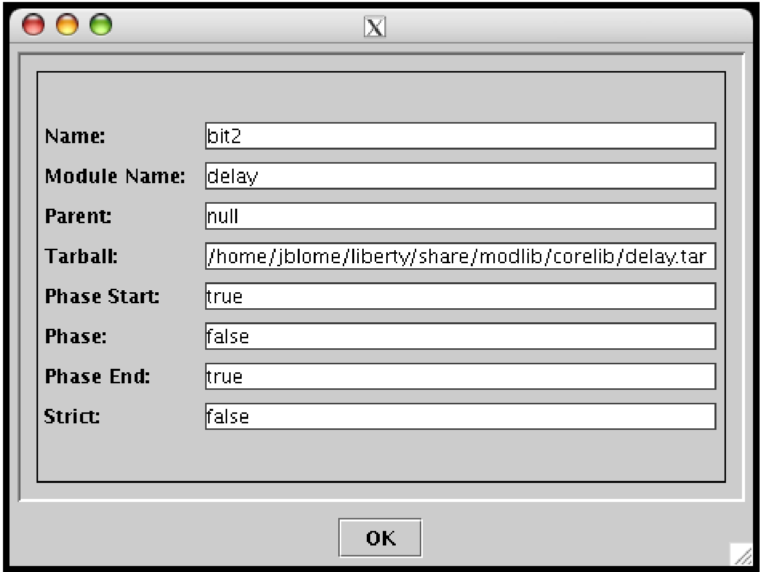


Figure 1-11. Instance Parameters Dialog



Customizing the Schematic View

Customization Primitives

The framework provided for drawing components on the canvas provides an interface for the user to convey both static and dynamic rendering information to the component. Static rendering information is conveyed to the component via `properties` and dynamic rendering information is conveyed via `commands`. We will discuss `commands` later in Chapter 2; the following is a brief description of how `properties` are used and stored.

Properties

Each canvas component defines a set of `properties` which it uses for the customization of it's display. The user can modify these properties by right clicking on a canvas component and clicking the menu item "View Visual Properties." A dialog listing some of these properties is shown above in Figure 1-10. These properties can be stored and reloaded if the user wishes by pressing the appropriate button in the schematic view window as demonstrated in the Section called *The Visualizer Schematic View Window*. The file containing these properties will be stored in the file: `lss_file_name.lss.properties`, and if a property file already exists, a backup will be stored in `lss_file_name.lss.properties~` before it is overwritten.

Open Issue

- Properties are not type checked in the current system, so writing code which assumes the wrong value type, or entering invalid data into a property editor dialog, may result in program errors.

The property file consists a series of key-value pairs, where the key is the full hierarchical name of the component concatenated with the property name and the value is a string consisting of the value type and the value. A brief example of the properties for the instance `bit0` follows:

Example 1-1. Sample Properties

```
1              bit0.Width=int 120
2              bit0.Height=int 62
3              bit0.Shape=string "Rounded Rectangle"
4              bit0.Label Font Size=int 14
```

Here, it is important to note, that a user may specify the default properties for every instance of a specific module type, by simply defining the module parameter `lvl_string` in the module definition. The user may, however, override these values on a per-instance basis, by simply providing a property file, or modifying the schematic view and storing the property file. It is important to note that in defining the properties in a the `lvl_string`, each property must end with a line break in order to be parsed. The same properties, could be defined as the default properties for the delay module as follows:

Example 1-2. Sample Properties

```
1              lvl_string = <<<
```

```
2               ${this}.Width=int 120
3               ${this}.Height=int 62
4               ${this}.Shape=string "Rounded Rectangle"
5               ${this}.Label Font Size=int 14
6             >>>;
```

Customizing the Visual Representation of Canvas Components

This section will discuss how the user may further customize the visual representation of canvas components and features of the schematic view by extending classes found in the canvas framework.

Customizing the Visual Representation of Instances

The canvas defines an extensible interface for defining canvas components. This framework defines two base types, the `SchematicFigure` and the `Drawable`. The SchematicFigure is a hierarchical element consisting of both subfigures and `Drawable` elements. The `Drawable` is an atomic element used to paint shapes and text on the canvas. The SchematicFigure interface is defined in the file: {VISUALIZER_SRC}/src/Liberty/visualizer/canvas/figure/SchematicFigure.java and the Drawable interface is defined in the file: {VISUALIZER_SRC}/src/Liberty/visualizer/canvas/drawable/Drawable.java. There are a number of abstract classes defined in order to ease the burden of implementing certain types of figures. The file: {VISUALIZER_SRC}/src/Liberty/visualizer/canvas/figure/lse/PluggableInstanceFigure.java defines the interface for rendering a figure that represents an LSS instance. Two implementations of this interface: DefaultInstanceFigure and GenericInstanceFigure exist in the same directory and may be used as the basis for defining custom rendering classes. Another implementation, the ALUInstanceFigure resides in the extensions directory.

The instance figures described above all define a property named `Display Class` which allows the user to specify the name of the class that should be used to render the instance representation. This class file must be available in the user's `CLASS_PATH` environment variable in order to be loaded. The example lss document used in this chapter is available in the visualizer source directory: {VISUALIZER_SRC}/samples/lfsr.lss and {VISUALIZER_SRC}/samples/lfsr.lss.properties, and makes use of all of the features discussed in this chapter.

Chapter 2. Dynamic Visualization of LSE Configurations

This chapter briefly describes the mechanisms through which a user of the visualizer may conduct interactive visualization of the execution of a binary simulator.

Basic Functionality

The visualizer interacts with the simulator via rpc calls made through a jni interface. All relevant files are located in the directory {VISUALIZER_SRC}/src/clp.

The SchematicFigure interface as described in the Section called *Customizing the Visual Representation of Instances* in Chapter 1 requires that every figure representing an LSS instance implement the function:

Example 2-1. SchematicFigure Interface Function

```
1        public void handleCommand(String command){}
```

Where the command can be any arbitrary string of text. The figure may choose to ignore the string or it may parse the string and carry out some actions accordingly. This mechanism may be used by the simulator to pass animation information on to a canvas element, and allows for a visualizer user to easily extend the animation facilities of a figure by simply extending its class and overriding the `handleCommand` function.

The DefaultInstanceFigure class discussed in the Section called *Customizing the Visual Representation of Instances* in Chapter 1 by default understands how to parse two basic commands. These commands are:

Example 2-2. DefaultInstanceFigure Commands

```
1       showTable(boolean value)
2       setValueAt(int col, int row, String value, int color)
```

Now, it is important to note that the DefaultInstanceFigure is designeed to render a widget representing a table of data. Thus, through the two commands listed above, the simulator can inform the DefaultInstanceFigure to display it's table and also to set the value at a particular location in the table. In order to do this, the simulator must be instrumented to call the rpc functions that will interact with the visualizer. An example of such code, taken from our lfsr example, would be as follow:

Example 2-3. Simulator Instrumentation

```
1       collector STORED_DATA on "bit2" {
2         record=<<<
3           char *value_string = malloc(40*sizeof(char));
4           snprintf(value_string, 40*sizeof(char), "setValueAt(0, 1, \"%d\", %d)",
5             *datap, 0xFF0000);
```

```
6             vis_server_handle_command("bit0", value_string);
7             vis_server_update_current_cycle(LSE_time_get_cycle(LSE_time_now));
8             free(value_string);
9           >>>;
10        };
11
```

Note that two function calls to the visulaizer server are made here. We will examine both of these functions. The prototype for the first function is:

Example 2-4.

```
1        int vis_server_handle_command(char *elementID, char *command)
```

This call made by the simulator makes an rpc call to the visualizer server, which will in turn, call the "handleCommand" function for the instance named by the parameter `elementID`.

The second function has the prototype:

Example 2-5.

```
1        int vis_server_update_current_cycle(int current_cycle)
```

This function is used to inform the visualizer that the current cycle has changed and it should update its display.

The results of executing the instrumented simulator can be seen in Figure 2-1 and Figure 2-2 shown below. The functionality of both the visualizer and simulator rpc servers can be increased by augmenting the files found in {VISUALIZER_SRC}/src/clp. Any changes made to these files will be linked directly into the simulator executable provided that the simulator is linked to the visualizer CLP as demonstrated in the Section called *The Visualizer Editor Window* in Chapter 1.

Figure 2-1. Execution Animation in the Schematic View

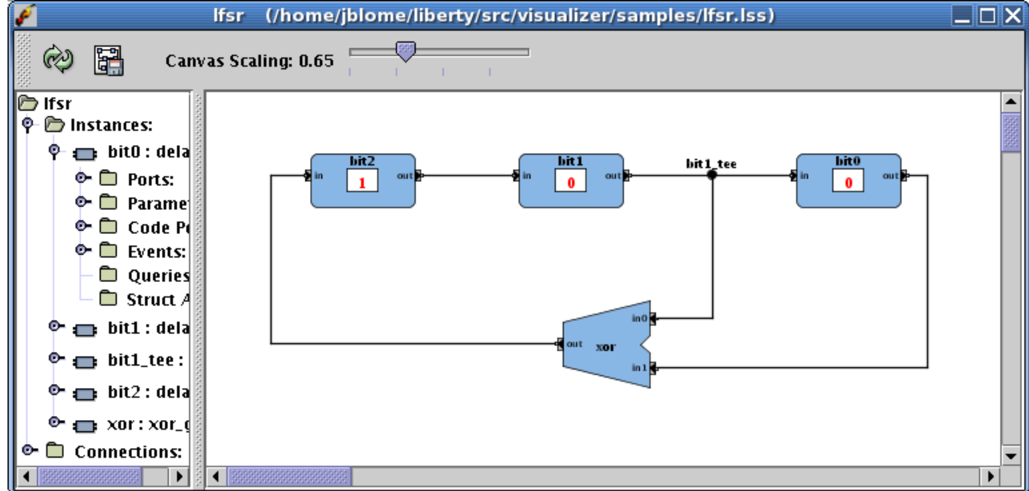


Figure 2-2. Execution Results