
Software Requirements Specification

for

V²HDL: Visual VHDL

Version 1.0

December 3, 2007

Prepared by:

Benjamin Burton, Iris Howley,
Jeff Patti, Thomas Wise

Drexel University

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Background	3
1.3	Scope	3
1.4	Definitions, Acronyms, and Abbreviations	3
1.5	Overview	4
2	Overall Description	5
2.1	Product Perspective	5
2.1.1	User Interfaces	5
2.1.2	Software Interfaces	5
2.2	VHDL source creation	5
2.2.1	Memory Constraints	5
2.3	Product Functions	5
2.4	User Characteristics	5
2.5	Requirements Apportioning	5
3	Specific Requirements	6
3.1	External Interface Requirements	6
3.2	Functional Requirements	6
3.2.1	File Manipulation	6
3.3	Components	6
3.4	Component Manipulation	6
3.5	Software System Attributes	7
3.5.1	Maintainability	7
3.5.2	Portability	7
3.5.3	Testability	7
3.6	User Interface	7
4	Non-Functional Requirements	17
4.1	Development	17
4.1.1	Versioning Control	17
4.1.2	Documentation	17
4.1.3	Graphical Interface	17
4.2	System Requirements	17
4.2.1	Java Runtime Environment Requirements	17
4.2.2	Symphony EDA Sonata Requirements	17
4.3	Future Work	17
5	Use Cases	18
5.1	Create a Visual VHDL Diagram of a Full-Adder from Scratch	18
5.2	Adding an Algorithmically-Generated Component to the Diagram	19
5.3	Edit an Existing Visual VHDL Diagram	19
5.4	Insert an Existing Visual VHDL Diagram into Working Diagram	19
5.5	Insert VHDL Code into Working Diagram	19
5.6	View a “Blackbox” Component in Visual VHDL	19
5.7	Convert Working Visual VHDL Diagram to VHDL Code	20
5.8	Clock	20

List of Figures

1	Possible toolbox GUI component for the application	7
2	Possible “Primitive” component section for the toolbox	8
3	Possible “import” blackbox section for the toolbox	9
4	Possible “generatable” section for the toolbox	9
5	Image of a possible menu bar for the application	10
6	Prototype of a possible toolbar for the application	10
7	View of the circuit logic design area contained within the Logic Diagram Editor window . . .	11
8	Pop-up menu provided when user right-clicks on a VHDL primitive component	11
9	Pop-up menu provided when user right-clicks on a blackbox or generated component	12
10	Pop-up menu provided when user right-clicks on a signal	12
11	A user selecting to split a signal	12
12	The effect of splitting a signal	13
13	A user beginning to drag the endpoint of a signal	13
14	A user finishing a drag to connect a signal to a component	13
15	A typical file access dialog box	14
16	Attributes section of Blackbox component properties dialog	15
17	Image section of Blackbox component properties dialog	15
18	Inputs/Outputs section of Blackbox component properties dialog	16
19	User Activities	18
20	A Gate Diagram of a Full Adder	18
21	Interaction between V ² HDL and Symphony EDA Sonata	20

1 Introduction

1.1 Purpose

This document specifies the software requirements for the V²hdl visual VHDL tool. The purpose of V² is to provide an educational tool for the development and understanding of high-level circuit logic diagrams and their hardware virtualization in the VHDL language.

The system allows for circuit logic diagrams to be generated from VHDL primitives (AND, OR, and NOT gates, etc.). Circuit logic may also be implemented using blackbox components, which have underlying logic encoded in VHDL generated dynamically or user-specified. This functionality is achieved via a drag-and-drop graphical user interface.

There is also an interface for the dynamic creation of high-level components such as multiplexors and n-bit arithmetic logic units. Users may specify the number of inputs and outputs for such components, whose underlying logic is developed algorithmically via plug-ins to the system.

1.2 Background

The final project for the course CS282: System Architecture II at Drexel University requires the development of a pipelined implementation of a MIPS processor in the VHDL language. Accomplishing this task presumes the students have a substantial understanding of the VHDL language. Learning VHDL consumes large quantities of time that would otherwise be spent on lectures or working on the project. The motivation for this application is to provide a framework for students to visually develop their pipelined processor implementation, without knowing the specifics of VHDL. This allows students to spend more time making design decisions related to their particular processor implementation instead of learning the particularities of the VHDL programming language. By providing an intuitive framework for development, students will require reduced training in application use, thereby freeing up teaching resources for more relevant aspects of systems architecture.

1.3 Scope

This document describes the software requirements of the V²HDL system. The intended audience of this document exclusively includes developers, testers, and end-users of V²HDL.

1.4 Definitions, Acronyms, and Abbreviations

API Application Programming Interface: a source code interface that an operating system or library provides to support requests for services to be made of it by computer programs.

Blackbox a technical term for a device or system or object when it is viewed primarily in terms of its input and output characteristics.

Circuit diagram a simplified conventional pictorial representation of an electric circuit. It shows the different components of the circuit as simplified standard symbols, and the power and signal connections between the devices.

Clock a clock signal is a signal used to coordinate the actions of two or more circuits. A clock signal oscillates between a high and a low state, normally with a 50% duty cycle, and is usually in the form of a square wave. Circuits using the clock signal for synchronization may become active at either the rising edge, falling edge, or both edges of the clock cycle.

Control unit part of a CPU or other device that directs its operation. The outputs of the unit control the activity of the rest of the device.

Dia free software/open source general-purpose diagramming software, developed as part of the GNOME project's office suite and was originally created by Alexander Larsson.

Dialog box a dialog box is a special window, used in user interfaces to display information to the user, or to get a response if needed.

GUI Graphical User Interface: a type of user interface which allows people to interact with a computer and computer-controlled devices.

Hotkey a key or set of keys that performs a predefined function.

Menu bar a region where computer menus are housed. Its purpose is to house window- or application-specific menus which provide access to such functions as opening files, interacting with an application, or help.

Microsoft Visio diagramming software for Microsoft Windows. It uses vector graphics to create diagrams.

MIPS processor Microprocessor without Interlocked Pipeline Stages: a RISC microprocessor architecture developed by MIPS Technologies.

Plug-in a computer program that interacts with a host application (a web browser or an email client, for example) to provide a certain, usually very specific, function "on demand".

Primitive data types provided by a programming language as building blocks. Primitive types are also known as *built in types* or *basic types*.

Reserved word a reserved word is a word which has a special grammatical meaning to a language and cannot be used as an identifier in that language.

Rubber-band box a method for selecting an area within the diagram. A rubber-band box involves clicking and holding down the mouse at a particular spot and stretching a box out from the point. When the mouse is released, all items contained within the box are selected.

Signal signals are often scalar-valued functions of time (waveforms), but may be vector valued and may be functions of any other relevant independent variable.

SVG Scalable Vector Graphics: an XML specification and file format for describing two-dimensional vector graphics, both static and animated.

Toolbar a row, column, or block of onscreen buttons or icons that, when clicked, activate certain functions of the program.

VHDL VHSIC hardware description language.

VHSIC A 1980s U.S. government program to develop Very-High-Speed Integrated Circuits.

1.5 Overview

The remainder of this document contains background information regarding the V²HDL system, as well as the functional and non-functional requirements of the system.

2 Overall Description

2.1 Product Perspective

2.1.1 User Interfaces

As the purpose of this application is to provide an educational tool for computer science students to develop MIPS processors with a minimal understanding of the VHDL language, learning how to use the application requires substantially less time than learning the VHDL programming language.

2.1.2 Software Interfaces

The application is easily expanded to add additional components not initially provided within the application.

2.2 VHDL source creation

The primary VHDL environment used by students to compile and simulate their VHDL code is the Sonata application provided by Symphony EDA. As a result, the program outputs human-readable VHDL source code compatible with this environment.

2.2.1 Memory Constraints

The application requires a maximum of 512 MB of RAM.

2.3 Product Functions

The V²HDL application provides the following functions:

- The user can make a graphical circuit logic diagram
- The user can export said diagram to VHDL
- The user can import previously created VHDL components
- The user can add layers of abstraction to their project by “blackboxing” components.

2.4 User Characteristics

The users of the V²HDL system should be familiar with applications such as Dia, Visio, or Adobe Illustrator type environments. Also, the user should be familiar with logic gates as this is a prerequisite for understanding and designing processors.

2.5 Requirements Apportioning

Priority	Description
1	Must be completed for v1.0
2	Should be completed for v1.1
3	Should be completed for v2.0
4	Not expected to be completed. Only to indicate where the software will go.

3 Specific Requirements

3.1 External Interface Requirements

0010 The application provides an interface such that other programmers will be able to add new algorithmically generated components to the system.

3.2 Functional Requirements

3.2.1 File Manipulation

0100 **Create New** - The system allows for the creation of a logic diagram which appears as a blank document from which the user may add various components. **Priority 1**

0110 **Open** - The system loads an existing logic diagram saved in a file on the system and provides a file dialog in order to determine the desired file. **Priority 1**

0120 **Save** - The user can save a logic diagram to a file on the system. This file is not necessarily human-readable. The system provides the user with a file dialog in order to determine the location to save the output file. **Priority 2**

3.3 Components

The focus of the application is to manipulate different circuit logic “components” to create a circuit logic diagram. V²HDL will allow use of these components:

0200 **VHDL Primitives** VHDL primitives are components of a system which are base level gate logic entities of a VHDL system. These components are similar to primitives of any object oriented system. Such components include, but may not be limited to AND, OR, NOT, NAND, NOR and XOR gates. **Priority 1**

0210 **User-Generated Blackbox Components** Blackbox components consist of any VHDL entity which is generated by the user. Such blackbox components may be specified by a given VHDL implementation file, a V²HDL file created within the application, or a placeholder for something the user wishes to implement at a later point in time. **Priority 2**

0220 **Algorithmically-Generated Components** These components of the system are those which are generated by a supplied algorithm. An algorithmically-generated component may be created by the user given a “plug-in” to the application. Such a plug-in allows the user to create a component based on parameters supplied by the user. **Priority 2**

0230 **Signals** Signals are VHDL datatypes which allow for the connection of components within a circuit logic diagram. Signals are analogous to a “wire” connecting two or more components. Signals may be split at designated points so that they serve as inputs to multiple components. **Priority 1**

3.4 Component Manipulation

Using a toolbox, the user selects a component and places it into the circuit diagram development space, giving the user the ability to add primary VHDL components, as well as blackbox or generated components.

0300 **Click-and-Click** The user clicks a component from the toolbox, and clicks in the circuit diagram development space to add a component to the diagram. **Priority 1**

0310 **Click-and-Drag** The user clicks a component from the toolbox and drags it into the circuit diagram development space to add a component to the diagram. **Priority 1**

0320 **Reserved Words Priority 2** When naming components, users are not allowed to name components any of the listed VHDL reserved words:

abs, access, after, alias, ali, and, architecture, array, assert, attribute, begin, block, body, buffer, bus, case, component, configuration, constant, disconnect, downto, else, elsif, end, entity, exit, file, for, function, generate, generic, group, guarded, if, impure, in, inertial, inout, is, label, library, linkage, literal, loop, map, mod, nand, new, next, nor, not, null, of, on, open, or, others, out, package, port, postponed, procedure, process, pure, range, record, register, reject, rem, report, return, rol, ror, select, severity, signal, shared, sla, sll, sra, srl, subtype, then, to, transport, type, unaffected, units, until, use, variable, wait, when, while, with, xnor, xor

0330 **VHDL Code Creation Priority 1** The system generates VHDL source code programmatically by analysis of the user-generated circuit logic diagram. Such VHDL code is human-readable, and is compilable and executable within the Symphony EDA Sonata VHDL environment.

3.5 Software System Attributes

3.5.1 Maintainability

0400 To facilitate maintainability the application is designed with a decoupled object model.

3.5.2 Portability

0500 As the system is written in Java, it is portable to any system capable of running a Java Virtual Machine (JVM). The source code provided by the application is runnable on any system capable of running the Sonata environment provided by Symphony EDA.

3.5.3 Testability

0600 The application is designed in such a manner that its usability is easily determined by effective unit tests, use case tests, and tests of the overall system functionality.

3.6 User Interface

The focus of this application is to provide an intuitive framework for visual VHDL design, and an intuitive graphical user interface is paramount to the usefulness of the application.

1000 **Toolbox Priority 1** The users adds components to a logic diagram via user interaction with a toolbar

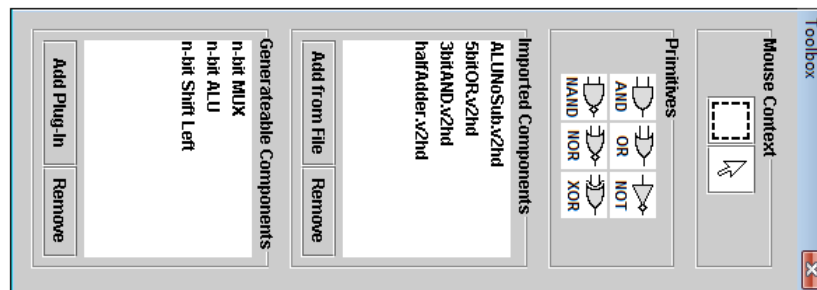


Figure 1: Possible toolbox GUI component for the application

which resides alongside a logic diagram editor window. This interface is similar with the interfaces in applications with similar purposes, such as Microsoft Office Visio and Dia. Using a similar interface for component addition provides intuitive access to commonly used system components, and reflects the goal of providing an easy to use application.

1100 **Logic Diagram Context Buttons Priority 1** As well as controlling the addition of components to a logic diagram, the toolbar is also responsible for switching the context of the mouse cursor functionality within the logic diagram. There are two contexts which are selectable:

1110 Individual Component Selection: The context for individual component selection allows the user to address a single component and its corresponding attributes through the use of a mouse click within the logic diagram editor. **Priority 1**

1120 Group Component Selection: The context for group component selection allows the user to address the properties of multiple components through the user of a **Priority 2** *rubber-band box* within the logic diagram editor.¹

1200 **Primitive Component Buttons Priority 1**

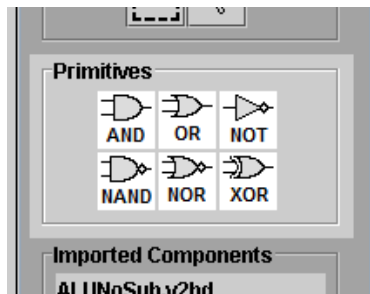


Figure 2: Possible “Primitive” component section for the toolbox

In order to add components to a logic diagram, the system contains buttons so the user can add VHDL primitives. These buttons may be utilized to add a primitive component in two distinct ways:

1210 Click-and-Click: The user clicks a primitive component within the toolbar, and then again within the logic diagram editor window to add a component to the logic diagram. **Priority 1**

1220 Click-and-Drag: The user clicks and holds down on a component within the toolbar. Releasing the mouse within the logic diagram editor adds the component to the logic diagram. **Priority 1**

1300 **Signal Button Priority 1** Since logic diagrams require individual components to be linked together with signals (i.e., wires), the toolbox provides functionality for the user to add such signals to the logic diagram editor. A “signal” button is in the toolbox, and allows the user to click-and-drag or click-and-click to add a signal as previously described.

1305 **Imported Blackbox Interface Priority 2** The toolbox interface also provides an interface to allow the user to add his own previously created logic diagrams as components to his current logic diagram. There are two separate types of import which are accessible through the interface:

1310 Component from VHDL: A blackbox component added from VHDL source in the form of either a *.vhd or *.vhdl file. **Priority 1**

1320 Component from V²HDL logic diagram: A blackbox component added from a preexisting Visual VHDL component created within the application. **Priority 2**

¹A rubber-band box is a method for selecting an area within the diagram. A rubber-band box involves clicking and holding down the mouse at a particular spot and stretching a box out from the point. When the mouse is released, all items contained within the box are selected.



Figure 3: Possible “import” blackbox section for the toolbox

The type of component created via the import interface is determined by the extension of the file selected for import. When the user wishes to import a component to the toolbox, he clicks the “Import” button in the imported component section. Selection of a valid component adds the component to the list of available components. From here, the component may be added to the circuit design area by click-and-drag or click-and-click for the item in the list (as described in the section on VHDL primitives).

1400 **Generated Component Interface Priority 3** The toolbox interface allows the user to add algo-

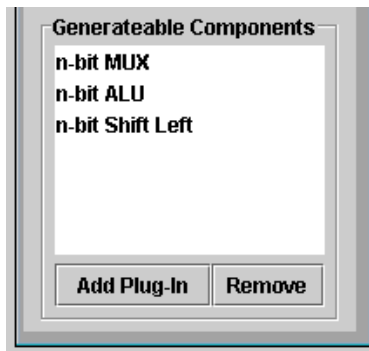


Figure 4: Possible “generatable” section for the toolbox

rithmically generated components via plug-ins to the application. Once the user selects a particular plug-in, a dialog window is displayed where the user enters data relevant to the component’s generation. The algorithm for generation, as well as the dialog used to obtain user input, is specified in the plug-in file. The plug-in adheres to a corresponding API for the application plug-in interface in order to properly achieve component generation. Plug-in files which violate these constraints produce an error dialog within the application.

1500 **Logic Diagram Editor Window Priority 1** The user makes modifications to his logic diagram through the logic diagram editor window. This window allows the user to make many manipulations to the positions of components within a logic diagram, as well as editing the particular attributes and relations associated with such items.

1510 **Menu Bar Priority 2** In order to provide an intuitive interface for logic diagram manipulation, the application has a menu bar with much of the functionality provided by applications with

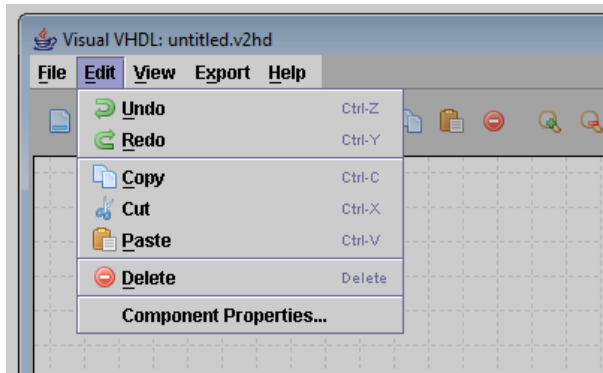


Figure 5: Image of a possible menu bar for the application

similar purposes. The following considerations were considered when developing the menu bar for the logic diagram editor:

- 1520 Commonly used menu titles are provided so that the user has a familiar experience using the application. The menu titles “File”, “Edit”, “View” and “Help” are used as they are commonly found in many applications. **Priority 2**
- 1530 Common menu items should be accessible via typical hotkey combinations provided in most applications. For example, the “Copy” function of the “Edit” menu should be accessible via the CTRL+C keyboard shortcut, “Undo” by CTRL+Z, etc. **Priority 3**
- 1540 Menu actions may have an associated icon, which presents a visual representation of the function of selection a particular action. **Priority 2**
- 1550 **Toolbar Priority 2** The logic diagram editor also contains a menu bar, which provides the most



Figure 6: Prototype of a possible toolbar for the application

commonly used functions related to creating a logic diagram within the application. The toolbar is designed with the following considerations:

- 1560 The icons in the toolbar provide a visual analogy of the action taken when clicking the button. **Priority 2**
- 1570 Tooltips are included with every icon in the menu bar so that the user is aware of the action that occurs when a button has been pressed. **Priority 2**
- 1580 Functionally related icons are separated into groups via whitespace. **Priority 2**

1700 **Circuit Logic Design Area** The logic design area is where the user views and manipulates the

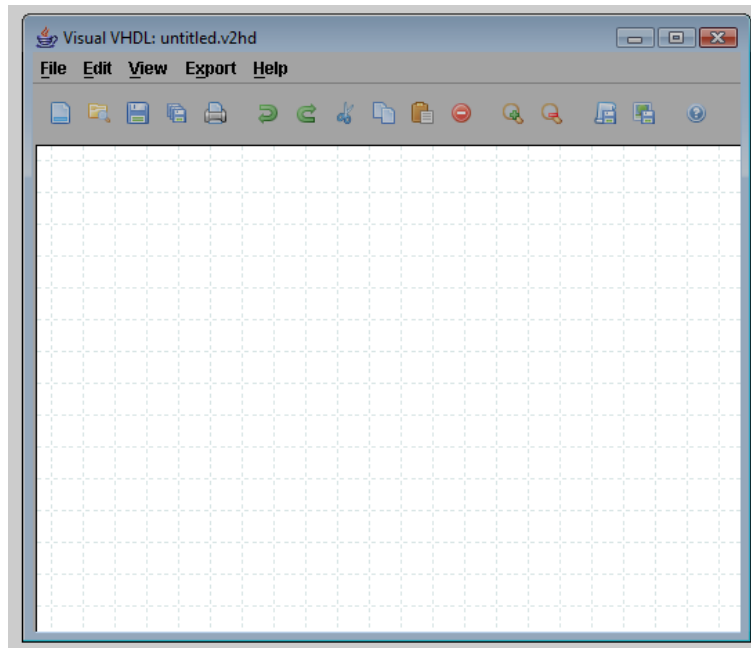


Figure 7: View of the circuit logic design area contained within the Logic Diagram Editor window

components related to the circuit logic diagram.

1800 **Right-Click Component Pop-up Menus** When the user right-clicks a component within the design area, he is presented with a pop-up menu. The contents of the pop-up menu varies based on the type of component being edited (e.g., primitive, blackbox, generated). Regardless of the type of component, the pop-up menu contains cut, copy, paste, delete and properties options.

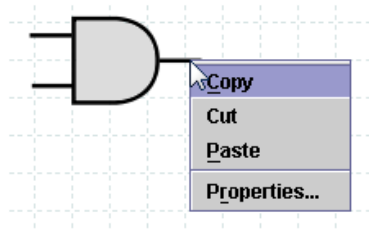


Figure 8: Pop-up menu provided when user right-clicks on a VHDL primitive component

1900 **Pop-up menu** The pop-up menu for blackbox and generated components contains an item to allow the user to view the VHDL code associated with the particular component. Additionally, the user may also select the option to open the V²HDL component associated with the component if it is available.

2000 **Signal Pop-up** The pop-up menu for a signal contains an item to allow the user to “split” the signal, as well as the other commonly used functionalities of cut, copy, delete and properties.

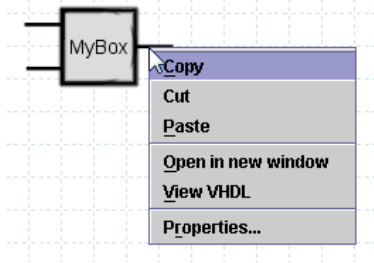


Figure 9: Pop-up menu provided when user right-clicks on a blackbox or generated component

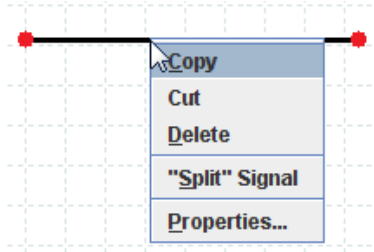


Figure 10: Pop-up menu provided when user right-clicks on a signal

2100 **Signal Splitting** Splitting a signal has the effect that the signal may be attached to the inputs of two separate components. This is analogous to splitting a wire so that has three endpoints.

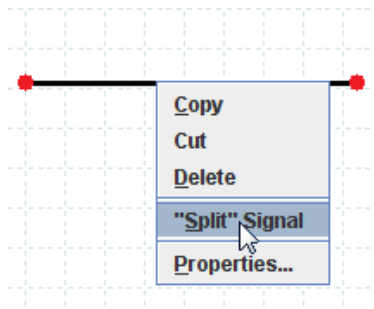


Figure 11: A user selecting to split a signal

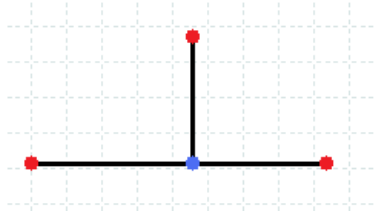


Figure 12: The effect of splitting a signal

2200 **Attaching Signals to Components** Signals may be attached to either the input or output of any particular component. This is mechanism the user utilizes to connect the components within the circuit logic diagram editor. Attaching signals to components is achieved by clicking a signal endpoint, and dragging it to a point on a component. The effect of this is that the signal will be attached to the component at the point at which the mouse is released.

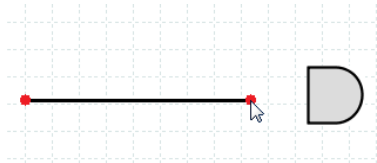


Figure 13: A user beginning to drag the endpoint of a signal

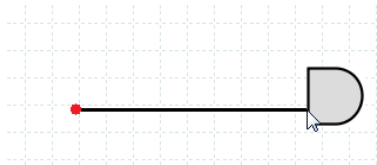


Figure 14: A user finishing a drag to connect a signal to a component

2300 **Application Dialog Boxes** The application provides a variety of different dialog boxes accessed in different contexts throughout the application.

2400 **File Access Dialog Boxes**

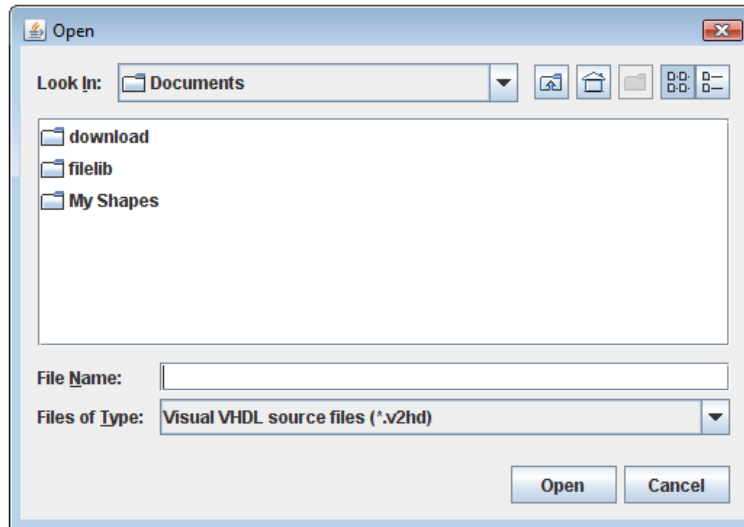


Figure 15: A typical file access dialog box

File access dialog boxes are provided for operations such as Open, Save, and Save As. These are standard dialog boxes which have various file filtering mechanisms in place depending on the type of file allowed. Valid file types for particular contexts are specified as follows:

- * Open: Valid V²HDL circuit logic diagram file types.
- * Save: Valid V²HDL circuit logic diagram file types.
- * Save As: Valid V²HDL circuit logic diagram file types.
- * Export to VHDL: *.vhd and *.vhd file types.
- * Export to Image: *.gif, *.jpg and *.png file types.
- * Import Component (Add from File in toolbox): Valid V²HDL circuit logic diagram file types, *.vhd or *.vhd
- * Generateable Component: Valid V²HDL component generator plug-in.

2500 **Primitive Component Properties Dialog** The properties dialog for a VHDL primitive component includes information about the name assigned to the component, as well as the type of VHDL primitive. Additionally, the properties dialog contains information about the input and output signals which are attached to the component.

2600 **Imported Blackbox Component Properties Dialog** The properties dialog for a VHDL imported blackbox component contains all the information specified in the primitive component properties dialog. Additionally, the blackbox component properties dialog contains a reference to the associated VHDL file if applicable. The associated file may be modified from within this dialog, or specified by the user via a file access dialog box.

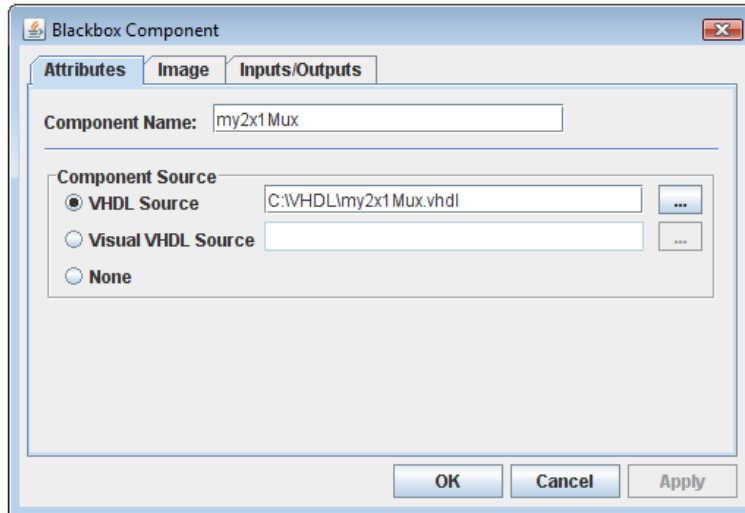


Figure 16: Attributes section of Blackbox component properties dialog

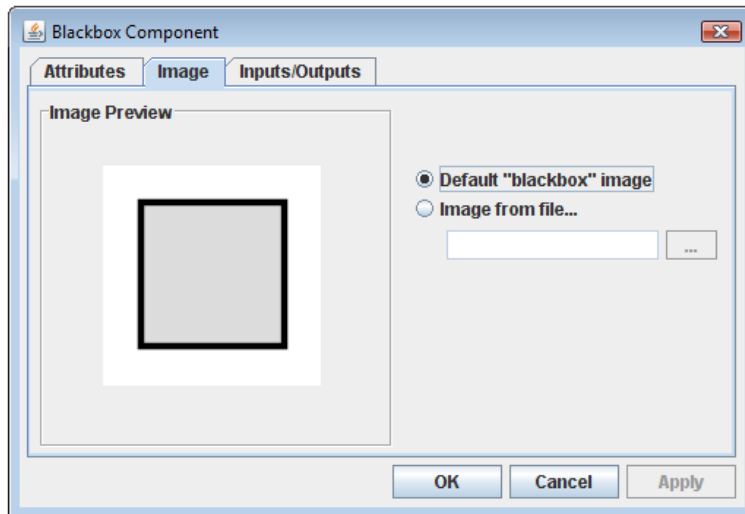


Figure 17: Image section of Blackbox component properties dialog

The properties dialog for a blackbox VHDL component also includes a mechanism for specifying the image associated with the component. The user selects either the default blackbox component image, or an SVG image specified by an input file dialog. The VHDL component properties dialog also contains a section for specification of the inputs and outputs to the component. Note that this is only editable in the context of a component which does not have a corresponding VHDL or V²DHL file. In these cases, the inputs and outputs will be visible, but not modifiable.

2700 **Generated Component Properties Dialog** The generated component properties dialog also contains all of the same information as the primitive component properties dialog. Additionally, this dialog contains a reference to the plug-in file which generated the component. As such, the dialog contains an option to modify the component properties via access to the plug-in file.

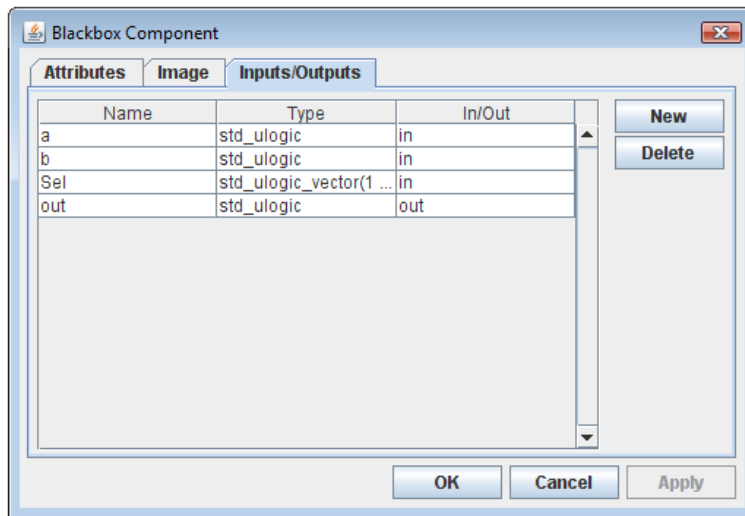


Figure 18: Inputs/Outputs section of Blackbox component properties dialog

4 Non-Functional Requirements

4.1 Development

4.1.1 Versioning Control

Code development will be controlled using using SVN versioning system.

4.1.2 Documentation

All external project documentation will be done in \LaTeX .

All internal documentation for code and API will be supplied via JavaDoc.

4.1.3 Graphical Interface

The graphical user interface for the application will be implemented using the Java Swing libraries. As such, the application will use the default Java Look and Feel, and not one that is operating system dependent in order to promote cross-platform interoperability.

4.2 System Requirements

4.2.1 Java Runtime Environment Requirements

As the application will be written in Java 1.6, an installation of Java Runtime Environment 6.0 or later will be a necessary component for any system on which the application is run. The requirements for JRE 6.0 are as follows:

4.2.2 Symphony EDA Sonata Requirements

The application will output VHDL compliant with Symphony EDA Sonata 3.1, which has the following requirements:

- Hardware platform: PC with Intel Pentium II equivalent or better.
- Video: 1280x1024 resolution or higher (lower resolutions will work, but are not recommended).
- Network Information Card (or Ethernet card) required for non-free (licensed, demo or purchased) versions.
- Operating System: Windows 2000/XP, or Linux.
- Memory: Minimum of 32MB available RAM.
- Disk Space: Minimum 40MB + additional space for vendor libraries.

4.3 Future Work

The application must be easily expandable for future specifications of the VHDL language, as well as future iterations of Symphony EDA.

5 Use Cases

The following is a list of use cases. There is only a single user considered, though each individual user may be categorized based upon skill (beginner, intermediate, advanced).

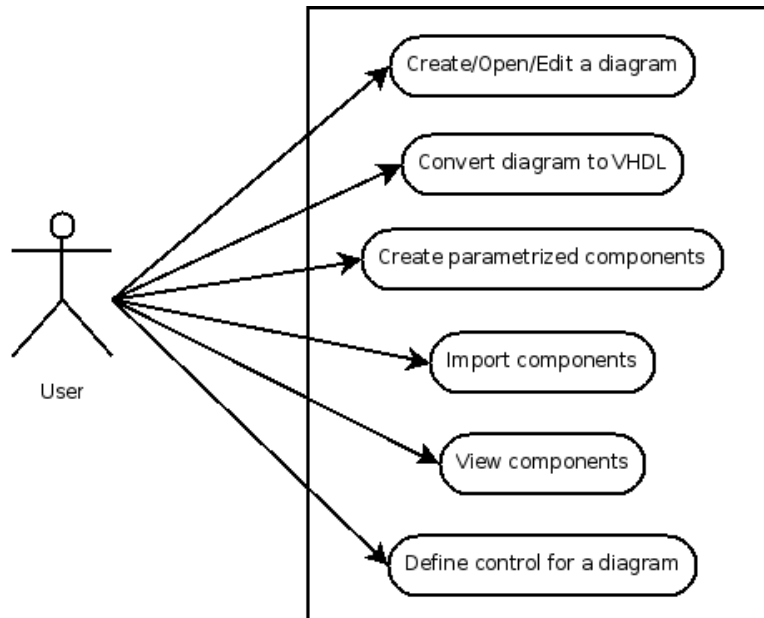


Figure 19: User Activities

5.1 Create a Visual VHDL Diagram of a Full-Adder from Scratch

The user wishes to use the Visual VHDL system to create a diagram of a full-adder from scratch.

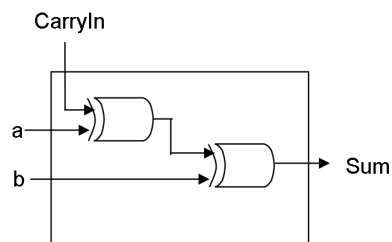


Figure 20: A Gate Diagram of a Full Adder

First, the user selects the creation of a “New” document. This is achieved by clicking on the menu bar icon for a new document, selecting New from the File menu, or using the CTRL+N hotkey combination. A new/blank canvas opens. Then, the user selects components from the toolbar’s VHDL primitives. The user clicks on the component and then clicks on the desired location on the canvas to place it. Alternatively, double-clicking the component pops up the options associated with that component (if any) and then places the component in the center of the canvas.

In this case, the user selects and places two XOR gates, and then proceeds to place connecting wires. The user selects the connector item from the menu, and then selects an output-location from a gate/component. As the user’s mouse moves to the input-location of another gate/component, the wire moves with the movement of the mouse until the user selects his input-location. If the user does not select an output-location or

an input-location, the wire ends/begins wherever the user clicks. The user may later drag the connector ends to a location at a later point in time. The user may remove a component or connector from the canvas by clicking on it and then pressing the delete key. The user may right click on either of the XOR components or the signals to provide names for them. If this is not done, the signals and XOR gates retain their default values.

The user may undo/redo a minimum of six actions, using either menu options or keyboard shortcuts.

5.2 Adding an Algorithmically-Generated Component to the Diagram

The user wishes to add a parameterized component to the VHDL diagram.

The user selects a parameterized component (such as an n-bit adder) from the toolbox/library (the usual double-click vs. single-click behavior applies). The menu that pops up with editing options appears, giving the user the option to describe what n-bit size to use. When the user is finished editing the options, he selects OK and the component is placed on the canvas.

5.3 Edit an Existing Visual VHDL Diagram

The user wishes to open an existing Visual VHDL diagram and modify it.

The user selects opening a document using the Open option from the File menu or clicking the Open icon on the toolbar. A file selection dialog appears. When the user selects his desired file, that files diagram opens and he edits the diagram, as described above.

5.4 Insert an Existing Visual VHDL Diagram into Working Diagram

The user wishes to insert an already created Visual VHDL diagram into another existing diagram.

The user looks at the “Imported Components” area of the toolbar. If the user has previously imported this component, he may drag-and-drop or click-and-click the corresponding item from the list. Otherwise, the user clicks the “Add from File” button within the “Imported Components” area. A file selection dialog appears, and the user selects the desired file. The user then uses drag-and-drop or click-and-click on the added list item. When the user selects the diagram to import, it appears as a “Blackbox” on the canvas, with input-locations and output-locations appropriate to the number in the actual diagram.

5.5 Insert VHDL Code into Working Diagram

The user wishes to insert VHDL code into an existing Visual VHDL diagram.

The user selects the option to create blackbox component, clicks the desired location on the canvas and a window pops up, allowing the user to input VHDL code, and specify the input and output signals in the code. The blackbox then appears where the user clicked on the canvas (alternatively, the user may always double-click a library component and the component will appear in the center of the canvas).

5.6 View a “Blackbox” Component in Visual VHDL

The user wishes to examine the details of a blackbox on the canvas.

The user right-clicks the blackbox on the canvas and a selects the “Open in new window” option from the displayed pop-up menu. If the blackbox is originally from a Visual VHDL diagram, the diagram will appear in the pop up window. The user may then double-click a blackbox within that view, but only one blackbox will ever be viewable at a single time. If the blackbox was originally VHDL code or a VHDL-modified diagram, then the blackbox view mode appears as the VHDL code used, with the specified input and output signals.

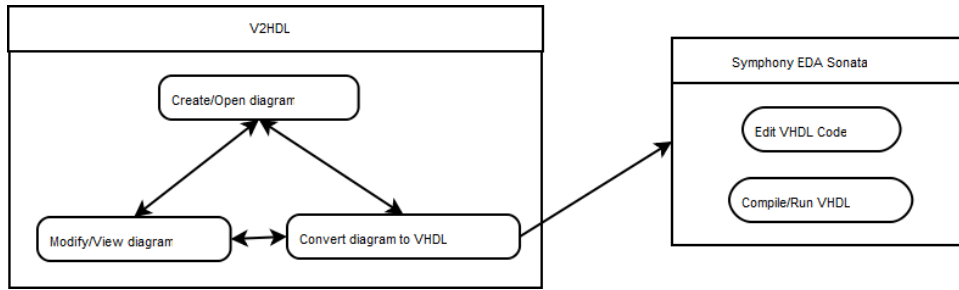


Figure 21: Interaction between V²HDL and Symphony EDA Sonata

5.7 Convert Working Visual VHDL Diagram to VHDL Code

The user wishes to convert the current Visual VHDL diagram into VHDL code.

The user selects the “Convert to VHDL” option from either the “Export” menu or the menu bar. A save file dialog opens, prompting the user to select a location and name to save the VHDL code.

5.8 Clock

The user wishes to add a clock to their project for timing

The user drags the clock primitive component and drops it on the canvas. They then can connect the clock to as many components as they wish. To adjust the rate of ticks, the user can right click on the clock and change its properties.