System-on-Chip Environment (SCE Version 2.2.0 Beta): Manual

Lukai Cai
Andreas Gerstlauer
Samar Abdi
Jerry Peng
Dongwan Shin
Haobo Yu
Rainer Dömer
Daniel D. Gajski

Technical Report CECS-TR-03-45
December 2003

System-on-Chip Environment (SCE Version 2.2.0 Beta): Manual

Lukai Cai
Andreas Gerstlauer
Samar Abdi
Jerry Peng
Dongwan Shin
Haobo Yu
Rainer Dömer
Daniel D. Gajski

Technical Report CECS-TR-03-45
December, 2003

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
+1 (949) 824-8919

http://www.cecs.uci.edu

Contents

List of Figures

System-on-Chip Environment (SCE Version 2.2.0 Beta): Manual

L. Cai, A. Gerstlauer, S. Abdi, J. Peng, D. Shin, H. Yu, R. Dömer, D. Gajski
Center for Embedded Computer Systems
University of California, Irvine

Technical Report CECS-TR-03-45
December, 2003

1 Introduction

The SCE provides an environment for modeling, synthesis and validation. It includes a graphical user interface (GUI) and a set of tools to facilitate the design flow and perform the aforementioned refinement steps. The two major components of the GUI are the Refinement User Interface (RUI) on the left and the Validation User Interface (VUI) on the right as shown in Figure 1. The RUI allows designers to make and input design decisions, such as component allocation, specification mapping. With design decisions made, refinement tools can be invoked inside RUI to refine models. The VUI allows the simulation of all models to validate the design at each stage of the design flow. Each of the boxes corresponds to a tool which performs a specific task automatically. A profiling tool is used to obtain the characteristics of the initial specification, which serves as the basis for architecture synthesis. The refinement tool set automatically transforms models based on relevant design decisions. The estimation tool set produces quality metrics for each intermediate models, which can be evaluated by designers. With the assistance of the GUI and tool set, it is relatively easy for designer to step through the design process. With the editing, browsing and algorithm selection capability provided by RUI, a specification model can be efficiently captured by designers. Based on the information profiled on the specification, designers input architectural decisions and apply the architecture refinement tool to derive the architecture model. If the estimated metrics are satisfactory, designers can focus on communication issues, such as protocol selection and channel partitioning.With communication decisions made, the communication refinement tool is used to generate the communication model. Finally, the implementation model is produced in the similar fashion. The implementation model is ready for RTL synthesis. We are currently in the process of developing tools for automating the synthesis tasks for system level design shown in the exploration engine. The manual concentrates on automatic synthesis process.

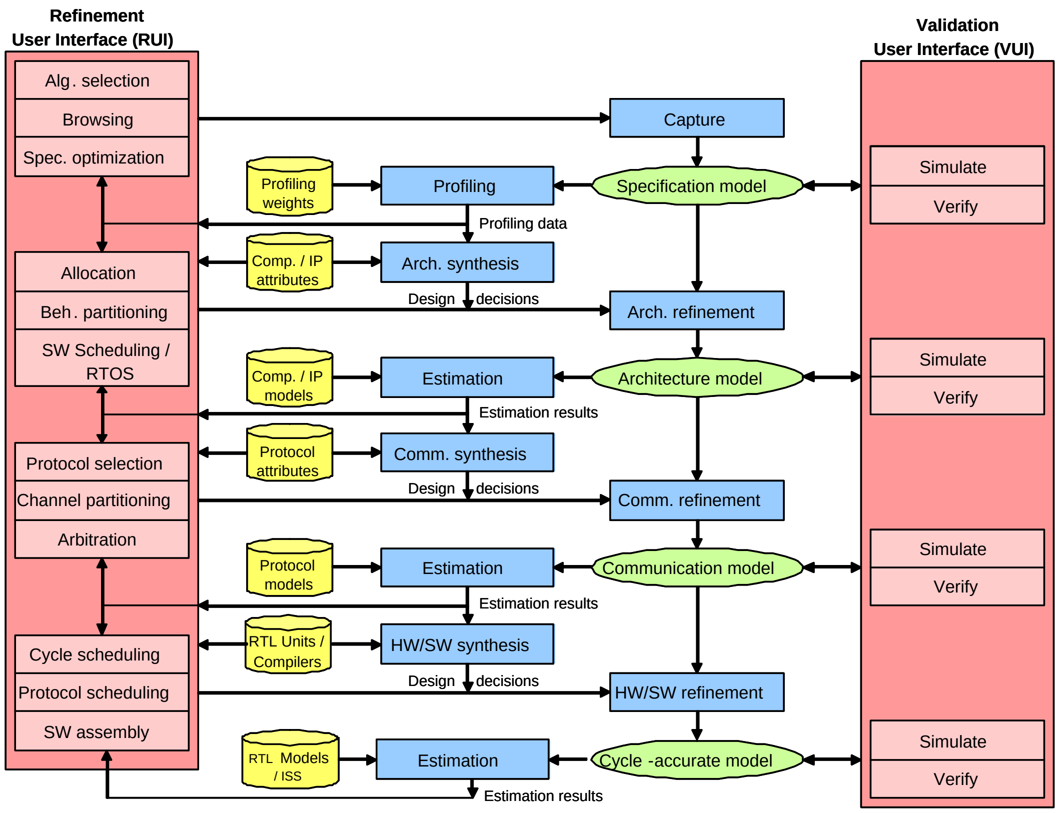


Figure 1: System-on-Chip Environment.

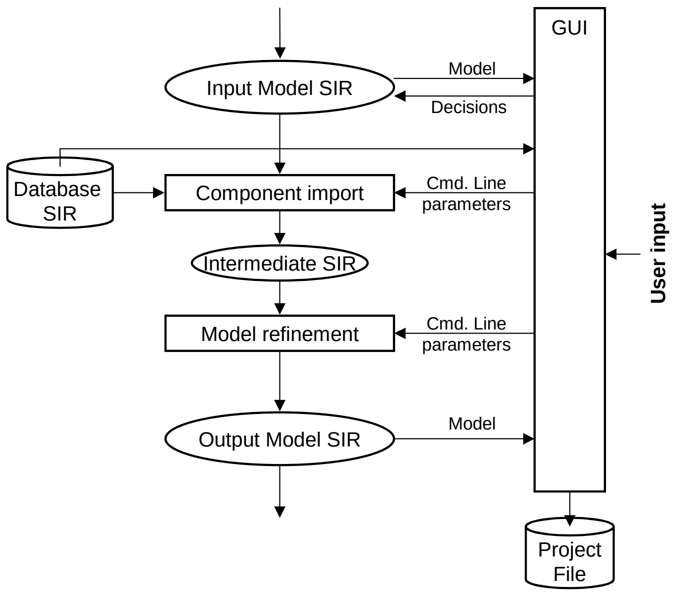


Figure 2: Overview of the software architecture of SCE.

2 Overview

Figure 2 shows an overview of the software architecture of each synthesis step of SCE. Each step requires an input model of the intended design in the form of valid SpecC code [1]. For example, the input model is the specification model for the architecture refinement, the architecture model for communication refinement.

Architecture synthesis reads information about available components from a database of processing elements (PEs). Communication synthesis reads information about available components from databases of buses. The database is read in binary form as a file in SpecC Internal Representation (SIR) format. Through SCE's graphical user interface (GUI), the user can browse the specification/architecture/communication model and PE/bus database, and the user is provided with a GUI to enter design decisions to come up with different design decisions. For architecture synthesis, the decisions are computation architectures and mappings of the specification onto those architectures. For communication synthesis, the decisions are communication architecture and mappings of the channels to buses.

For each candidate design decision as determined by the user, each step automatically generates an output model that exactly reflects the design decisions made. The output model is the architecture model for architecture synthesis, and the communication model for communication synthesis.

Information about design models and their relationships are tracked by the SCE and can be stored in project files that can be read and written by SCE in a custom XML format. Generated architecture models along with the corresponding project file(s) can then be passed to the following tools in the SCE design flow.

Internally, the SCE consists of separate components for component import, architecture/communication refinement, and graphical user interface (GUI). The main SCE application is the SCE GUI which in turn calls import and refinement components as needed. Component import and refinement are command line tools that are called and executed by the GUI where the GUI supplies the correct command line parameters, captures the output and handles (normal or abnormal) results. The GUI reads and displays design models, lets the user browse the database, and provides facilities to enter design decisions. Design decisions are stored by the GUI as annotations in the input design model. For generation of output design models, the GUI first passes the annotated design to the component import tool which, as requested by the GUI, imports the necessary component models out of the database into the design and writes an intermediate design model. The GUI then calls the refinement tool on the intermediate design model. The refinement tool in turn generates the final output architecture model which can be read, displayed and browsed through the GUI.

The run time of automatic model generation is in the order of minutes. Therefore with SCE, users can rapidly experiment with dozens, even hundreds of alternative design decisions to make the optimal design decisions.

2.1 Modules

The block diagram and flow chart of the SCE is displayed in Figure 3. The blocks represent SCE modules where each block is associated with one or more GUI elements. GUI elements and hence SCE blocks can be further classified into three types: the Main Window, input dialogs and display windows.

Besides above three types of windows, there are two types of arrows in Figure 3: solid arrows and dashed arrows. Solid arrows represent the pop-up relations. The window at the arrow head will be triggered by clicking a button or selecting a menu item in the window at the arrow tail. For example, by clicking *Open* button in *File* column in the menu of window Main Window, users will trigger *File Open* dialog. The text that appears on some arrows represents the arrow's task name. On the other hand, dashed arrows represent information updates. The data in the window at the arrow head will be updated by clicking a button in the window at the arrow tail. In order to simplify Figure 3, the arrows for error and

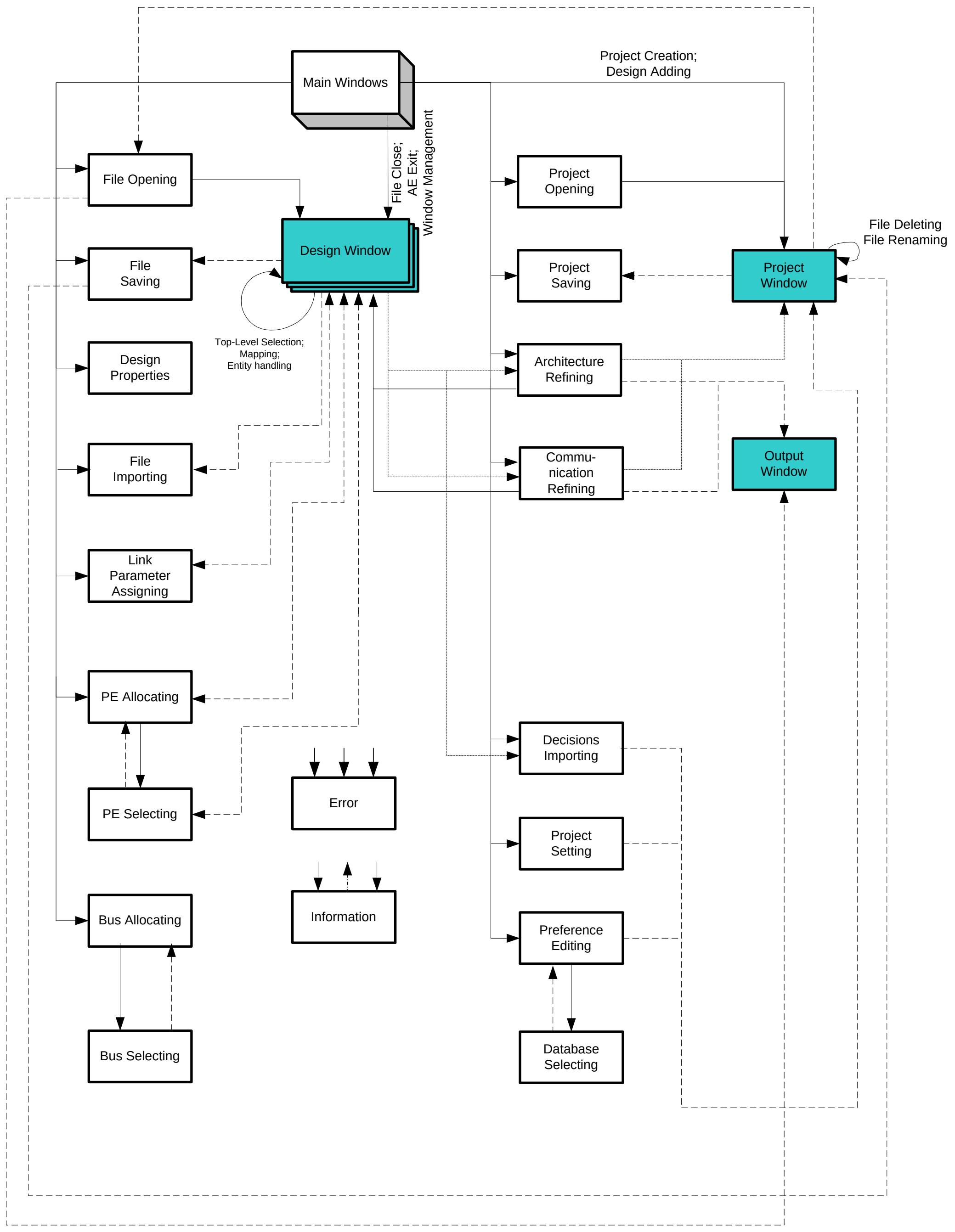


Figure 3: Module diagram of SCE.

information dialogs are not connected. They are popped up when the error or information occurs, which will be described in Section 4.

2.1.1 Main Window

The main window (shown as 3-D block in Figure 3), is the default, top-level window of the GUI and hence the SCE application. The main window contains a menu bar and a status bar through which the application can be controlled by the user and through which feedback is provided to the user about the state of the application.

2.1.2 Input Dialogs

Input dialogs (shown as regular blocks) allow users to input the design or project decisions by selecting, editing, or typing the details. They are all pop-up dialogs. Users directly/indirectly pop up all the input dialogs by clicking buttons or selecting menu items. Input dialogs include:

(a) File Open dialog for selecting the design file which users want to open.

(b) File Save dialog for saving current file as a new file with the specified name.

(c) Design Property dialog for displaying the information related to the process of SCE, such as command history and processing status of SCE.

(d) File Import dialog for selecting the design file which users want to import.

(e) PE Allocation dialog for adding PEs to and removing PEs from the design.

(f) PE Selection dialog for selecting PEs from the PE library (for input dialog PE Allocating).

(g) Bus Allocation dialog for adding busses to and removing busses from the design.

(h) Bus Selecting dialog for selecting busses from the bus database.

(i) Error dialogs for displaying of error messages of design tasks.

(j) Information dialogs for providing informational feedback to the user.

(k) Project Open dialog for reading a new project from disk.

(l) Project Save dialog for saving current project as a new project on disk with the specified name.

(m) Architecture Refinement dialog for selecting sub-tasks for architecture refinement.

(n) Communication Refinement dialog for selecting sub-tasks for communication refinement.

(o) Decision Import dialog for import the design decisions from other opened designs.

(p) Project Settings dialog for setting project preferences.

(q) Edit Preferences dialog for setting SCE preferences.

(r) Database Selection dialog for selecting the preferred database.

2.1.3 Display Windows

Display windows (shown as shaded windows in Figure 3) graphically display the information of files, projects, and the process status of SCE. Display windows are sub-windows under the main window. Display windows include:

(a) Design Window displays the contents and the attributes of the designs saved in the opened files. The displayed information includes hierarchy of behaviors, execution sequence of behaviors, and variable/port details of the selected behavior.

(b) Project Window displays the project information such as hierarchy of design models and included source files.

(c) Output Window displays the captured output of the SCE command line tools called by the GUI.

2.2 Requirements

The SCE requires a system with the following runtime platform:

Host machine Intel-compatible x86 PC, 500MHz or higher, recommended minimum 128MB of RAM, 200MB of free hard disk space.

Operating system RedHat Enterprise Linux WS, Version 3

In order to compile the source code of SCE, the following additional software packages have to be installed (in binary form including necessary header files):

- SIR library, Version 2.2.x (UC Irvine)
- Qt library and toolkit, Version 3.3.x (Trolltech, Inc.)
- PyQt library, Version 3.11 or higher (Riverbank Computing, Ltd.)

Note that in order to redistribute the libraries together with the compiled SCE, commercial licenses of the above tools have to be obtained as necessary.

2.3 Interfaces

SCE interfaces can be separated into internal interfaces for information exchange between SCE components and external interfaces for information exchange with other tools both within SCE and outside of SCE.

2.3.1 Internal Interfaces

Components inside SCE exchange information through design models, command line parameters, logging output, and exit codes.

Design models Design models are exchanged between SCE components in the form of SpecC files stored on any file system supported by the underlying operating system (Linux). In addition to the SpecC code for the design models themselves, SCE components exchange information via annotations for design decisions and design meta-information.

Command line parameters When calling command line tools, the SCE GUI will pass information for controlling the tool in the form of command line parameters.

Logging output Command line tools will produce logging output during execution. This logging output is captured by the GUI and displayed to the user in the GUI's output window.

Exit codes Command line tools signal status information (success or error codes in the case of tool failure) to the GUI in the form of their exit codes. The GUI analyzes command line tool exit codes and translates them into necessary information or error messages.

2.3.2 External Interfaces

The SCE exchanges information with other tools and with the designer via design models, databases, project files and via SCE's graphical user interface:

Design models Design models are exchanged between SCE's tools in the form of SpecC source code stored as files in any file system supported by the underlying operating system (Linux). SpecC source files are stored as text files in DOS or Unix end-of-line format where SCE will be able to read both formats and to export files in DOS form. Generally, SpecC source code is stored in plain ASCII format. However, SCE will be able to transparently handle Kanji-encoded comments and strings.

SCE will generally be able to import any valid SpecC code that is parsable according to the syntax and grammar of SpecC 2.0 (based on standard ANSI C) as defined in the SpecC Language Reference Manual (LRM), Version 2.0 [1]. Note that inside SCE no object code or executables are ever created and therefore models imported into SCE can include foreign code that depends on libraries outside (i.e. does not have to be linkable on) the underlying host platform (Linux). In contrast, since SCE will preprocess the SpecC source files on the host platform and inline any included code in its exported models, pre-prepared, clean header files with all specific code to be included have to be supplied together with the SCE import SpecC models as necessary for external tools.

In addition, individual models imported into SCE will have to conform to the specific rules and guidelines defined in the specification documents for each type of model.

Databases Databases are generated by the library builder and stored as a collection of binary SpecC Internal Representation (SIR) files managed via a specific file system hierarchy on top of the general underlying file system. SCE components then read component models from these SIR database files.

Project files Project files are stored as XML files in any file system supported by the underlying operating system (Linux). Project XML files are text files using Unix text file format. Project files use a custom XML format that is common to all tools in the SCE environment, i.e. the project XML file format is shared among the SCE tools and any SCE tool is able to read, write and modify project files generated by or used as input to any other SCE tool.

User interface All user input is entered in SCE through a graphical user interface (GUI). The SCE GUI is built on top of the X11 windowing system and as such can be run on any local or remote X window server.

2.4 Performance

The SCE will guarantee that for a typical design with less than 10,000 lines of code, less than 10 PEs, and less than 100 behaviors, variables and channels, automatic generation of the refined model will take less than 5 minutes.

3 Windows/GUI

The primary GUI of SCE is the Main Window, which is displayed in Figure 4. The Main Window consists of six parts:

(a) A Menu Bar that contains several columns of commands. Each column is a drop-down menu (see Section 3.1).

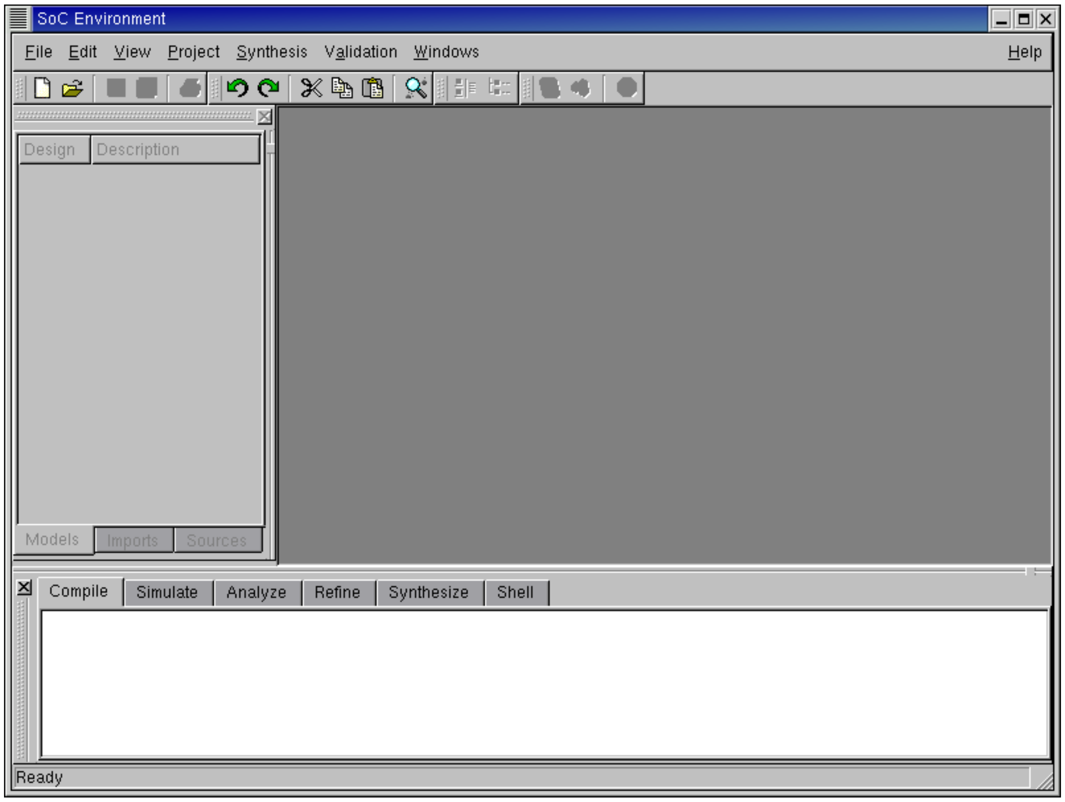


Figure 4: Main Window of SCE.

(b) A Tool Bar that contains a list of short-cut icons. Each icon represents a command in the menu bar.

(c) A Project Window (see Section 3.2).

(d) A Workspace that contains a number of opened Design Windows (see Section 3.3).

(e) An Output Window (see Section 3.4).

(f) A Status Bar that displays the current status of SCE, such as "Loading. . . " or "Ready".

In this section, we introduce organization-related and display-related details of Menu Bar, Project Window, Display Windows, and Output Window. Some windows contain drop-down menus or pop-up menus. The menus further contain design commands. The usage and functionality behind the commands will be described later in Section 4.

3.1 Menu Bar

The Menu Bar contains seven main menu entries: *File*, *Edit*, *View*, *Project*, *Synthesis*, *Validation*, and *Windows*. Each main menu entry is a drop-down menu which contains a number of commands. In general, unless otherwise noted, selecting a main menu entry will apply the corresponding action to the currently active design, i.e. to the design window in the workspace that currently has the input focus. If there is no currently active design window, menu commands will silently fail (do nothing).

3.1.1 File Menu

The *File* menu contains eight commands:

Open Selecting *Open* will allow loading and opening of an existing design file (see Section 4.3.1).

Close Selecting *Close* will close the currently active design (see Section 4.3.3).

Save Selecting *Save* will save the currently active design file (see Section 4.3.2).

Save As Selecting *Save As* will save the currently active design as a new file (see Section 4.3.2).

Save All Selecting *Save All* will save all the opened files (see Section 4.3.2).

Import Selecting *Import* will import a design file into the currently active design (see Section 4.3.4).

Properties Selecting *Properties* will display the properties of the currently active design (see Section 4.3.5).

Exit Selecting *Exit* will exit from and quit SCE (see Section 4.3.6).

3.1.2 Edit Menu

The *Edit* menu contains six commands:

Undo Selecting *Undo* will undo the previous action.

Redo Selecting *Redo* will redo the previous action.

Cut Selecting *Cut* will cut the selected text and save it in the buffer.

Copy Selecting *Copy* will copy the selected text and save it in the buffer.

Paste Selecting *Paste* will paste the content in the buffer to the place where the mouse points to.

Preferences Selecting *Preferences* will allow viewing and modifying of application preferences (see Section 4.1.1).

3.1.3 View Menu

The *View* menu contains four commands:

Source Selecting *Source* will display the source file of the selected design.

Graph Selecting *Graph* will display profiling graphes of the selected design.

Connectivity Selecting *Connectivity* will display the connectivity of behaviors of the selected design.

Hierarchy Selecting *Hierarchy* will display the behavior hierarchy graphically.

3.1.4 Project Menu

The *Project* menu contains seven commands:

New Selecting *New* will create a new project and open it (see Section 4.2.1).

Open Selecting *Open* will open an existing project file (see Section 4.2.2).

Close Selecting *Close* will close the current project (see Section 4.2.10).

Save Selecting *Save* will save the current project file (see Section 4.2.3).

Save As Selecting *Save As* will save the current project as a new project file (see Section 4.2.3).

Add Design Selecting *Add Design* will add the current design into the project.

Settings Selecting *Settings* will allow viewing and modifying of project settings (see Section 4.2.4).

3.1.5 Synthesis Menu

The *Synthesis* menu contains following commands:

Show Variables Selecting *Show Variables* will toggle displaying of variables in the currently active design window (see Section 4.4.4).

Show Channels Selecting *Show Channels* will toggle displaying of channels in the currently active design window. (see Section 4.4.4)

Allocate PE Selecting *Allocate PE* will allow allocation and selection of PEs/memories from the PE database (see Section 4.5.1).

Allocate Busses Selecting *Allocate Busses* will allow users to allocate busses from the bus database. (see Section 4.5.2).

Import Decisions Selecting *Import Design* will allow importing of design decisions from another opened design (see Section 4.5.3).

Architecture Refinement Selecting *Architecture Refinement* will perform architecture refinement to generate the refined architecture model (see Section 4.5.1).

Communication Refinement Selecting *Communication Refinement* will perform communication refinement to generate the communication model (see Section 4.5.2).

3.1.6 Validation Menu

The *Validation* menu contains four commands:

Compile Selecting *Compile* will compile the current design.

Simulate Selecting *Simulate* will simulate the current design.

Profile Selecting *Profile* will profile the current design.

Estimate Selecting *Estimate* will estimate the current design.

3.1.7 Windows

The *Windows* menu contains six commands:

Close Selecting *Close* will close the currently active design window in the Workspace.

Close All Selecting *Close All* will close all design windows in the Workspace.

Next Selecting *Next* will switch focus to and raise the next design window in the Workspace.

Previous Selecting *Previous* will switch focus to and raise the previous design window in the Workspace.

Tile Selecting *Tile* will rearrange the design windows in the Workspace in a tiled style.

Cascade Selecting *Cascade* will rearrange the design windows in the Workspace in a cascaded style.

Project Manager Selecting/unselecting *Project Manager* will display or undisplay the Project Window.

Output Window Selecting/unselecting *Output Window* will display or undisplay the Output Window.

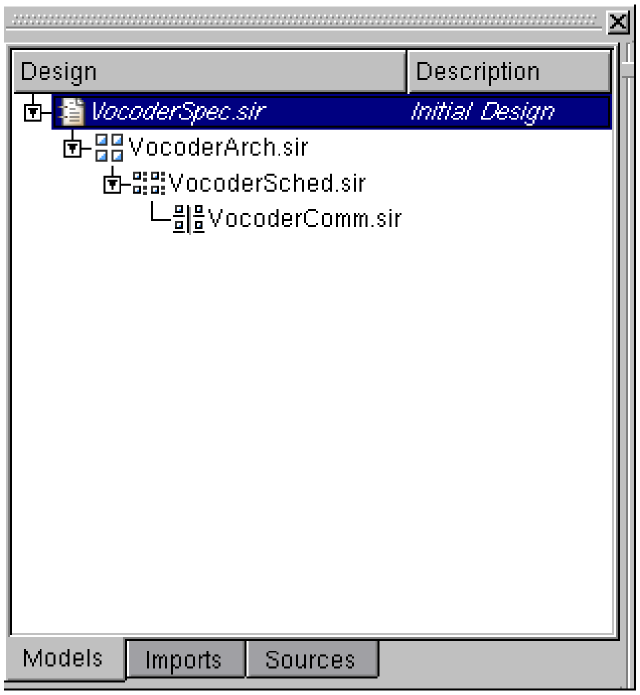
At the bottom of the *Windows* menu, the names of all opened design windows will be listed. Selecting the name of a design will switch focus to and raise (bring to the front) the corresponding design window.

Details about the usage and functionality of these commands are introduced in Section 4.6.

3.2 Project Window

The Project Window is a sub-window of the Main Window, displaying project information. In general, a project holds meta-information about a set of design files and their relationship, e.g. a parent-child relationship in case a design was generated from another design through refinement. Hence, each file represents a design of the project at one abstraction level. In addition, the project can hold project-specific settings for the design environment, such as the compiling and parsing environment (paths).

At any time, the SCE application can keep one project open and active in memory. The name of this currently active project is displayed in the title bar of the Main Window. The Project Window displays the hierarchical information of design files in the currently active project. If there is no active project, the Project Window is disabled.



VocoderSpec.sir Initial Design
VocoderArch.sir
VocoderSched.sir
VocoderComm.sir

Figure 5: Project Window (Models tab).

The Project Window can be detached or docked. Users can drag the window (by its title bar or handle) to the desired place. If the Project Window is detached, it can be floating and displayed anywhere on the desktop. If the Project Window is docked, it has to be attached to any of the borders of the Main Window.

The Project Window contains three tabs: *Models*, *Imports*, and *Sources*. By clicking the tab at the bottom of the window, the corresponding tab will be activated and brought to the front.

3.2.1 Models Tab

Figure 5 shows the screenshot of the Models tab. The Models tab contains two columns. Column *Design* displays the name of each design file in the project. If one design is derived or generated from another, the derived design is displayed as the child design of the previous one. The derivation hierarchy is indicated through connecting lines of a tree structure.

Users can select a design by clicking the row in which the design is displayed. Double-clicking on a row will open the corresponding design. If the design is currently opened and loaded, double-clicking will activate (switch focus to and raise to the front) the corresponding design window in the Workspace (see Section 3.5).

The Column *Description* shows an optional description of the design. By clicking into the column, the user can edit the description text of the selected design directly in the corresponding table cell.

Right-clicking on a row will open a context menu pop-up for the selected design. The context menu contains four commands:

Open Selecting *Open* will open the selected file (see Section 4.3.1). If the selected design is already opened, the corresponding design window in the workspace will be activated.

Delete Selecting *Delete* will remove the selected file from the project and optionally delete the file on disk (see Section 4.2.7).

Rename Selecting *Rename* will rename the selected file (see Section 4.2.8).

Change Description *Change Description* will trigger editing of the description of the selected file. (see Section 4.2.9)

Note that renaming and changing of description actions can also be triggered by clicking into the corresponding column of the selected file.

3.2.2 Imports Tab

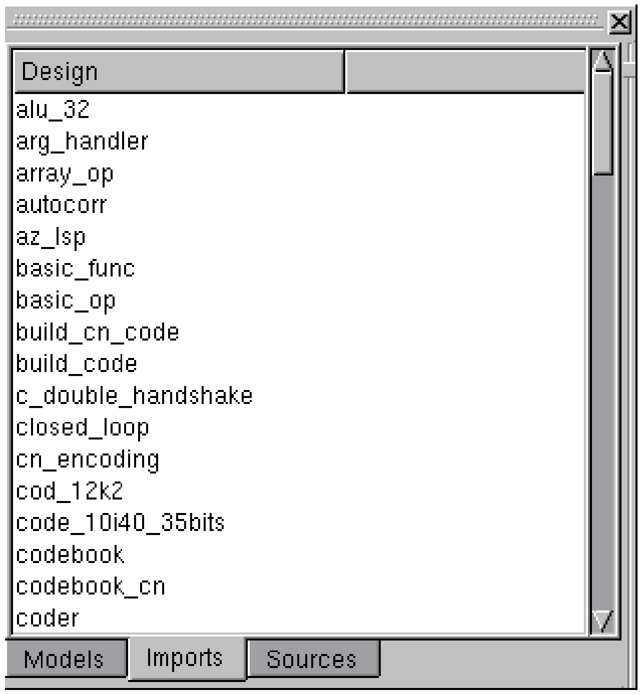
Figure 6 shows the screenshot of Imports tab. The Imports tab displays a lists of imported design names. The list contains the union of all the sub-designs that have been imported directly or indirectly into any of the design files that are part of the project.

Users can select an imported design by clicking the row in which the imported design is displayed. Double-clicking on an imported design opens the corresponding design in the workspace.

Right-clicking on an imported design opens a context-menu pop-up for the selected design. The context menu contains one command:

Open Selecting *Open* will open the import design file (see Section 4.3.1).

Note that the opening action is equivalent to double-clicking on the imported design.



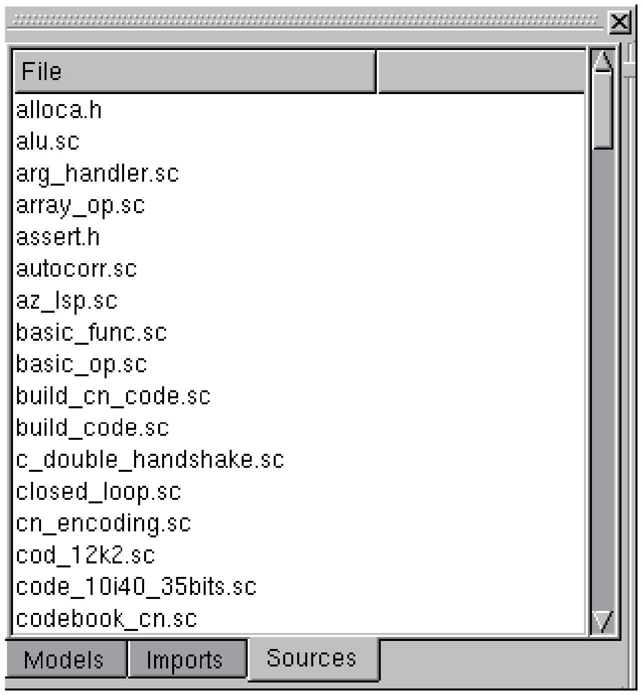
Design	
alu_32	
arg_handler	
array_op	
autocorr	
az_lsp	
basic_func	
basic_op	
build_cn_code	
build_code	
c_double_handshake	
closed_loop	
cn_encoding	
cod_12k2	
code_10i40_35bits	
codebook	
codebook_cn	
coder	

Figure 6: Project Window (Imports tab).

3.2.3 Sources Tab

Figure 7 shows the screenshot of Sources tab. The Sources tab displays a lists of names of source files. The list contains the union of all SpecC source files that are sources for the design files that are part of the project.

The Sources tab does not allow any action on the files and is for informative purposes only.



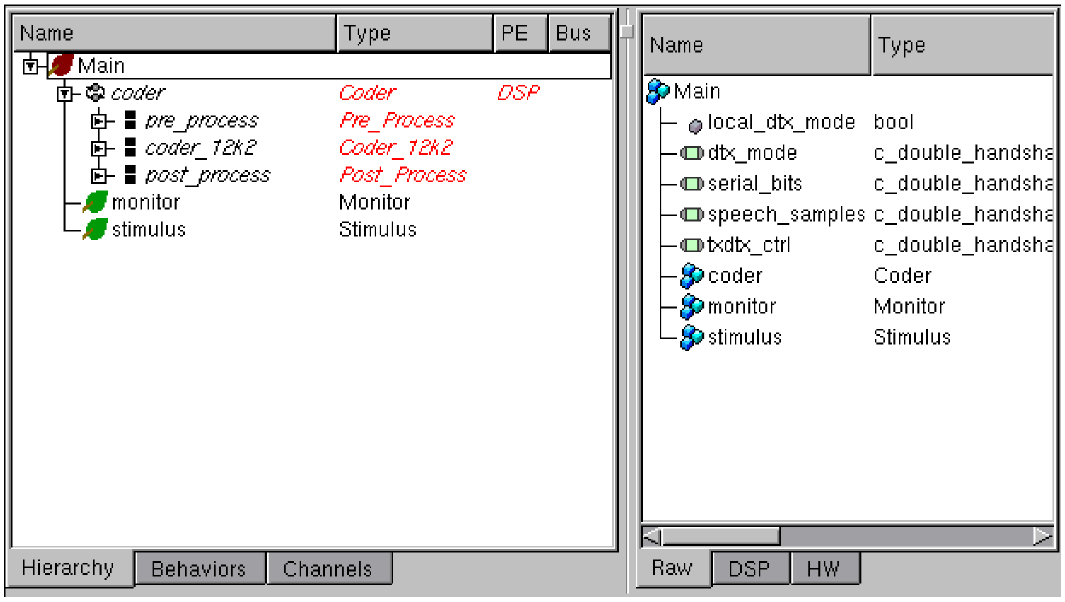
File

alloca.h
alu.sc
arg_handler.sc
array_op.sc
assert.h
autocorr.sc
az_lsp.sc
basic_func.sc
basic_op.sc
build_cn_code.sc
build_code.sc
c_double_handshake.sc
closed_loop.sc
cn_encoding.sc
cod_12k2.sc
code_10i40_35bits.sc
codebook_cn.sc

Figure 7: Project Window (Sources tab)

.

3.3 Design Window



Name	Type	PE	Bus
Main			
coder	*Coder*	*DSP*	
pre_process	*Pre_Process*		
coder_12k2	*Coder_12k2*		
post_process	*Post_Process*		
monitor	Monitor		
stimulus	Stimulus		

Name	Type
Main	
local_dtx_mode	bool
dtx_mode	c_double_handsha
serial_bits	c_double_handsha
speech_samples	c_double_handsha
txdtx_ctrl	c_double_handsha
coder	Coder
monitor	Monitor
stimulus	Stimulus

Figure 8: Design Window (Hierarchy tab).

The Design Window displays the content and the attributes of an opened design, and it allows browsing and navigation of the design hierarchy. The screenshot of the Design Window is displayed in Figure 8.

The Design Window consists of two parts: the side bar and the view pane. The side bar displays the basic information for navigation of the structure of the design. It further consists of three tabs: *Hierarchy*, *Behaviors*, and *Channels*. The view pane displays the detailed information for the entity that has been selected in the side bar.

3.3.1 Hierarchy Tab

The Hierarchy tab in the side bar are illustrated in Figure 8. In the column *Name*, the design hierarchy of behaviors, channels (optional) and variables (optional) is displayed in a tree form. The hierarchy tree indicates the entity type (variable, channel or sequential, parallel, FSM, or leaf behavior) visually through icons in the *Name* column. Entities are sorted according to their calling order in the design, i.e. according to their execution order in case of sequential or FSM behavior compositions. Displaying of channels and variables in the Design Window Sidebar Hierarchy tab can be toggled via the View menu and the toolbar (see Section 4.4.4).

At the root of the hierarchy tree in the Hierarchy tab, behavior (and optionally channel) types/classes are listed. At lower levels of the hierarchy, sub-behavior (and optionally channel and/or variable) instances inside the respective parent behavior (or channel) are listed. For any but the roots of the hierarchy tree, the column *Name* shows the name and the column *Type* shows the type of the respective design instances. For classes at the root of the tree, the *Name* column shows the name of the respective class (i.e. the type) and the *Type* column is empty.

For example (Figure 8), behavior class *coder* contains three child behavior instances *pre_process* (of type *Pre_Process*), *coder_12k2* (of type *Coder_12k2*), and *post_process* (of type *Post_Process*), which are executed in the fsm style identified by the symbol which has two balls connected by round arrows.

If PE allocation information is available, the Hierarchy tab contains a column *PE* which shows the PE mapping information. If no PE allocation information is available, the *PE* column is not shown. If the PE mapping is empty, this implies that the behavior in this row is mapped to the same PE to which its parent is mapped.

Right-clicking on a design entity row in the Hierarchy tab opens a context menu pop-up for the selected entity. The context menu contains four commands:

Rename Selecting *Rename* will rename the selected entity (see Section 4.4.1).

Delete Selecting *Delete* will delete the selected entity (see Section 4.4.3).

Change Type Selecting *Change Type* will allow plug-and-play to change the type of the selected entity. *Change Type* is not shown for the roots of the hierarchy tree.

Set as Top Level Selecting *Set as Top Level* will set the selected behavior as the top level design behavior (see Section 4.5.1).

3.3.2 Behaviors Tab

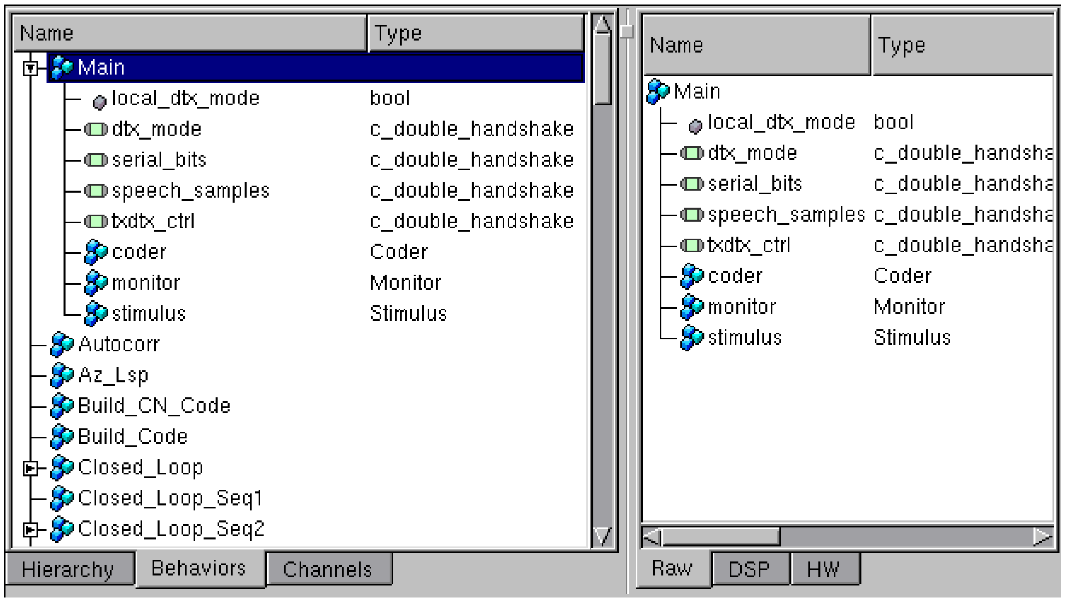
The Behaviors tab in the sidebar (Figure 9) lists all behavior types/classes in the design. The name of each behavior type/class is shown in a column *Name*. Behavior types are sorted by name and the sort order can be toggled by clicking into the *Name* column header.

Right-clicking on a behavior row in the Behaviors tab opens a context-menu pop-up for the selected behavior class. The context menu of the *Behaviors* tab contains three commands:

Rename Selecting *Rename* will rename the selected entity (see Section 4.4.1).

Delete Selecting *Delete* will delete the selected entity (see Section 4.4.3).

Set as Top Level Selecting *Set as Top Level* will set the selected behavior as the top level design behavior (see Section 4.5.1).



Name	Type
Main	
local_dtx_mode	bool
dtx_mode	c_double_handshake
serial_bits	c_double_handshake
speech_samples	c_double_handshake
txdtx_ctrl	c_double_handshake
coder	Coder
monitor	Monitor
stimulus	Stimulus
Autocorr	
Az_Lsp	
Build_CN_Code	
Build_Code	
Closed_Loop	
Closed_Loop_Seq1	
Closed_Loop_Seq2	

Name	Type
Main	
local_dtx_mode	bool
dtx_mode	c_double_handsha
serial_bits	c_double_handsha
speech_samples	c_double_handsha
txdtx_ctrl	c_double_handsha
coder	Coder
monitor	Monitor
stimulus	Stimulus

Figure 9: Design Window (Behaviors tab)

.

3.3.3 Channels Tab

In the Channels tab (Figure 10), all the channel types/classes in the design are listed. The name of each channel type/class is shown in a column *Name*. Channel types are sorted by name and the sort order can be toggled by clicking into the *Name* column header.

Right-clicking on a channel row in the Channels tab opens a context-menu pop-up for the selected channel class. The context menu of the *Channels* tab contains two commands:

Rename Selecting *Rename* will rename the selected entity (see Section 4.4.1).

Delete Selecting *Delete* will delete the selected entity (see Section 4.4.3).

3.3.4 View Pane

When clicking on a design entity row in any of the sidebar tabs, the corresponding row is selected and details of the selected entity are displayed in the view pane. The details shown are the contents of the respective entity and they include contained sub-entities including ports, methods, and variable, channel and child behavior instances. Both the name and the type of sub-entities are displayed in the *Name* and *Type* columns of the view pane, respectively. Elements of the view pane list are sorted by class (port, variable, behavior,

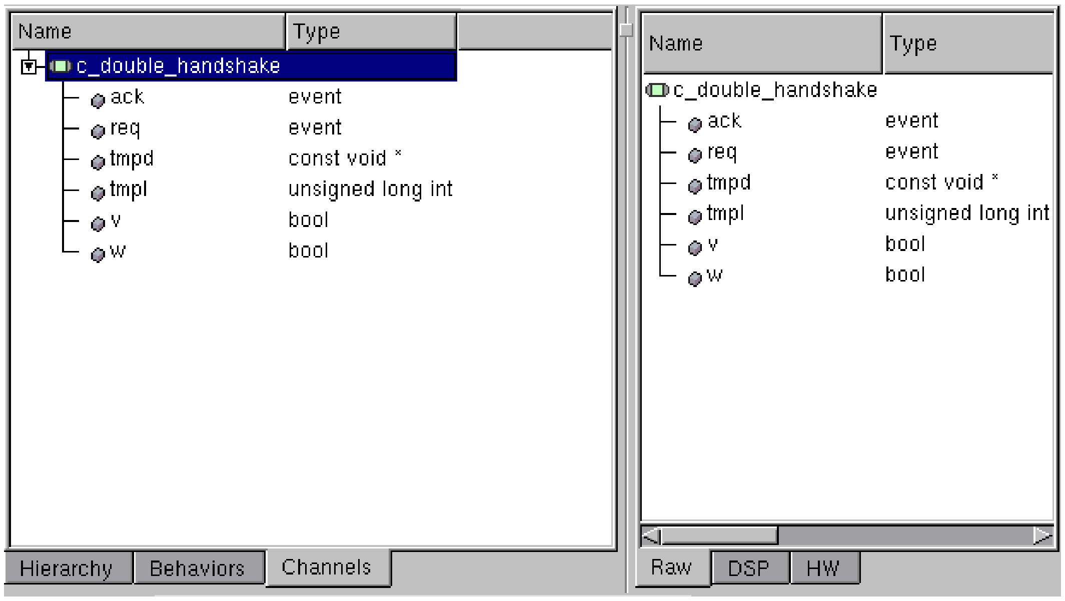
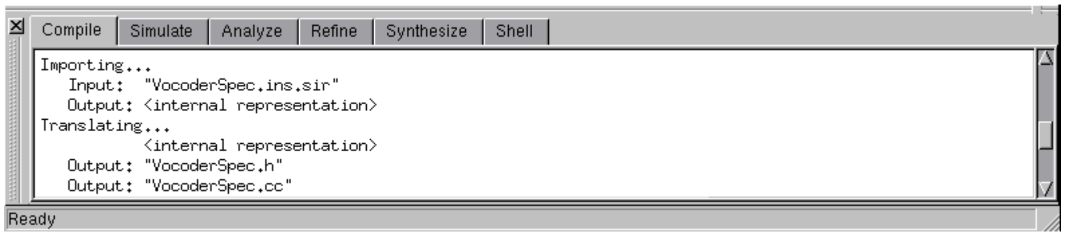


Figure 10: Design Window (Channels tab).

channel). Within each class and among classes, the sort order can be set by clicking on the *Name* or *Type* column headers.

3.4 Output Window



```
Importing...
   Input:  "VocoderSpec.ins.sir"
   Output: <internal representation>
Translating...
           <internal representation>
   Output: "VocoderSpec.h"
   Output: "VocoderSpec.cc"
```

Figure 11: Output Window

.

The Output Window displays the information related to the process of SCE, such as logged status, diagnostic and error output of background commands. The screenshot of Output Window is displayed in Figure 11. The Output Window contains six tabs: *Compile*, *Simulate*, *Analyze*, *Refine*, *Synthesize*, and *Shell*. For example, the *Compile* tab displays the log messages generated during preprocessing and parsing of SpecC code when opening, loading and importing design files. The *Refine* tab displays the log messages generated by the command line tools spawned by the main application GUI during design refinement. The Output Window is for informational purposes only and doesn't contain any button, box or context menu that users can click or edit.

The Output Window can be detached or docked. Users can drag the window (by its title bar or handle) to the desired place. If the Output Window is detached, it can be floating and displayed anywhere on the desktop. If the Output Window is docked, it has to be attached to any of the borders of the Main Window.

3.5 Workspace

In general, multiple designs can be open and loaded in the SCE application. The design windows for all currently opened and loaded designs are shown in the Workspace. Within the Workspace, design windows can be minimized, maximized, resized and closed freely via their title bar, title bar icons and handles on their window frames. Closing a design window closes the corresponding design (file).

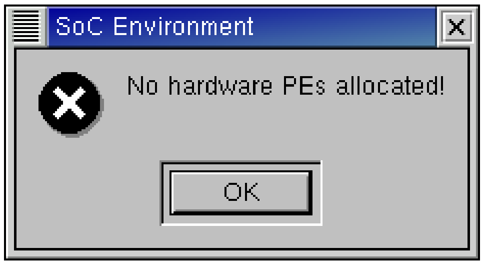
At any time, there is exactly one active design window in the Workspace. The active window is the one that has the input focus and it is visualized by highlighting its title bar. Unless otherwise noted, all menu, toolbar or other commands apply to the currently active design window. Clicking into a design window activates the corresponding window and raises it to the front of the Workspace. A newly opened design windows automatically becomes the active window.

3.6 Message Boxes

As a result of certain actions, the SCE application will pop up message box dialogs for feedback to or input from the user about handling of special situations. Message boxes are used to provide informative messages and to ask simple questions. In general, there are two types of message boxes: error dialogs and information dialogs.

3.6.1 Error Dialogs

If the application encounters an abnormal error situation in which user notification about the failure of the initiated action is required, an Error dialog will be popped up (Figure 12). The Error dialog displays an error message at the top-half of the Error dialog. At the bottom-half, an Error dialog contains one button: *Ok*. Clicking *Ok* will close the Error dialog and original dialog (if any) that prompted the message. After clicking, the original action that prompted the message is aborted and cancelled.



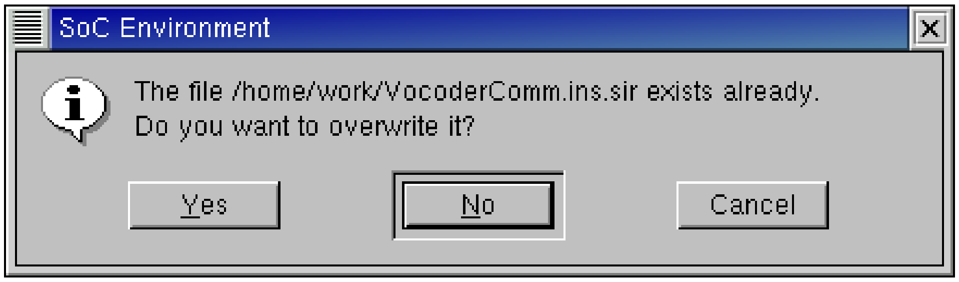
No hardware PEs allocated!

Figure 12: Error dialog

.

3.6.2 Information Dialogs

If the application encounters an abnormal situation in which user notification is required and the user is given several choices on how to continue, an Information dialog will be popped up (Figure 13). An information message and associated question is displayed at the top-half of the dialog. The bottom-half of the dialog contains three buttons: *Yes*, *No*, and *Cancel*. Clicking *Yes* will accept the recommendation and do the corresponding action. Clicking *No* will not accept the recommendation and will not do the corresponding action but will continue the original action that prompted the message in the first place. Finally, clicking *Cancel* will not do the recommended action and will also cancel the original action that prompted the message. Clicking one of above three buttons will close the Information dialog and original dialog (if any) that prompted the message.



The file /home/work/VocoderComm.ins.sir exists already.
Do you want to overwrite it?

Figure 13: Information dialog

.

4 Functionality

The functionality of SCE can be classified to six categories: application, project handling, file handling, design-entity handling, synthesis, and window management.

In this section, sub-windows or sub-menus described in Section 3 are referred using the following format: $Win : Sub$. *Win* refers to display windows:

- *Main* represents the Main Window.
- *Project* represents the Project Window.
- *Design* represents the Design Window.

Sub refers to drop-down menus or sub windows (tabs):

- For the Main Window, *Sub* is either *File*, *View*, *Project*, *Synthesis*, or *Windows* (drop-down menus introduced in Section 3.1).
- For the Project Window, *Sub* is either *Models*, *Imports*, or *Sources* (tabs introduced in Section 3.2).
- For the Design Window, *Sub* is either *Hierarchy*, *Behaviors*, or *Channels* (sidebar tabs introduced in Section 3.3).

For example, $Project :: Models$ refers to the Models tab in the Project Window.

Main menu or context menu commands described in Section 3 are referred to using the following format: $Win :: Sub \Rightarrow Command$ where *Command* refers to a command. For example, *Main::File⇒Open* refers to the *Open* command in the *File* menu of the Main Window menu bar. On the other hand, *Project::Models⇒Open* refers to the *Open* command in the context menu of the Project Window Models tab.

4.1 Application

The main application of SCE supports a set of persistent application settings. Application settings are persistently stored across different invocations of the tool. In fact, application settings are shared among all tools in the SCE environment, i.e. they are persistent across invocation of different tools at different times.

Application settings are stored in both system-wide and user-specific locations. System-wide application settings affect all users of SCE applications on the system. They are stored in a file on disk in a location that is configurable during compile time of SCE. User-specific application settings, on the other hand, are stored in a file in the user's Linux home directory. The application first reads the system-wide and then the user-specific settings, i.e. user-specific settings can override (if given) system-wide settings and if no user-specific settings are given, application settings default to the system-wide settings. If no system-wide settings are available, compiled-in defaults are used.

Application settings in general provide the standard settings (paths, etc.) to use by default for the different parts of SCE applications. Note that application settings can be overwritten or extended by project-specific settings (see Section 4.2). Application settings include:

Compiler settings A set of options for preprocessing and parsing SpecC source files. When opening/loading or importing a design file, the SpecC compiler (‘`scc`’, see Appendix A.1) is used internally to compile the SpecC source file into SCE’s internal SpecC Internal Representation (SIR) [2] format. Via the compiler settings, the options for preprocessing and parsing passed to the SpecC compiler are specified. Specifically, compiler settings contain the following:

Standard include path An ordered list of directories in which to search for include files during preprocessing.

Standard import path An ordered list of directories in which to search for imported files during parsing.

Macro defines An ordered list of preprocessor macro definitions.

Macro undefines An ordered list of preprocessor macro undefines.

Compiler options Additional compiler switches passed literally to the SpecC compiler. Possible compiler switches are switches for setting warning and verbosity levels.

Database paths Location of the database SIR files for the PE database, the CE database, and the bus database.

All paths in the application settings are relative to the current working directory when starting the application, i.e. relative paths in the settings are converted into absolute paths by prepending the working directory during startup of the application.

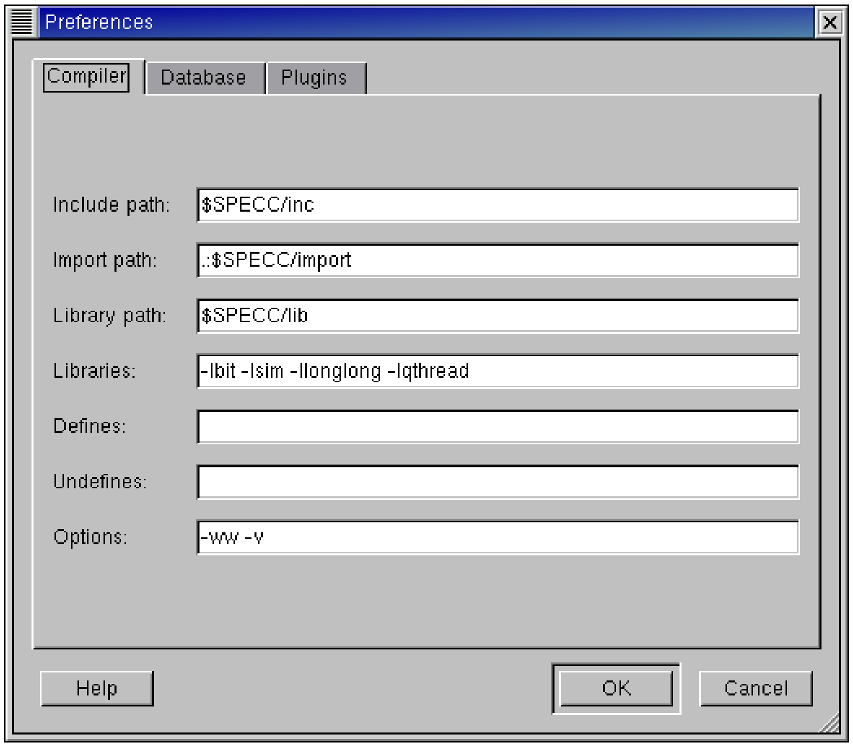
In terms of application settings, SCE supports functions to view and edit application settings/preferences.

4.1.1 Preferences Editing

Operation Preference editing allows viewing and setting of the application settings of SCE. Users start editing the preferences of SCE by selecting $Main :: Edit \Rightarrow Preference$. This will pop-up the Edit Preferences dialog, which allow users to browse and specify the compiler and database settings. The Edit Preferences dialog is illustrated in Figure 14.

There are three tabs in the Edit Preferences dialog: Compiler, Database, and Plugins. By clicking the tab at top-left corner of the window, users can select either of them for viewing and editing.

(a) The Compiler tab allows viewing and editing of compiler settings. The screenshot for Compiler tab is shown in Figure 14. The Compiler tab contains line edit boxes



Include path: $SPECC/inc

Import path: .:$SPECC/import

Library path: $SPECC/lib

Libraries: -lbit -lsim -llonglong -lqthread

Defines:

Undefines:

Options: -ww -v

Figure 14: Edit Preferences dialog (Compiler tab).

for all compiler settings. The text in the *Include Path* and *Import Path* lines defines the directory lists (separated by colons) for the standard include and standard import paths, respectively. The text in the *Defines* and *Undefines* lines define the list of macro defines and undefines (separated by semicolons), respectively. Finally, the text in the *Options* line defines the compiler options/switches.

(b) The Database tab allows for viewing and selecting of database file paths. The screenshot for Database tab is shown in Figure 15. Users can type in the file names and paths of PE, Bus and RTL databases in *PE Database*, *Bus Database*, and *RTL Database* line edit boxes. Besides typing in the databases file names, users can also select the names by using *Select* buttons next to the edit boxes. Clicking *Select* button will pop up a Database Selection dialog displayed in Figure 16

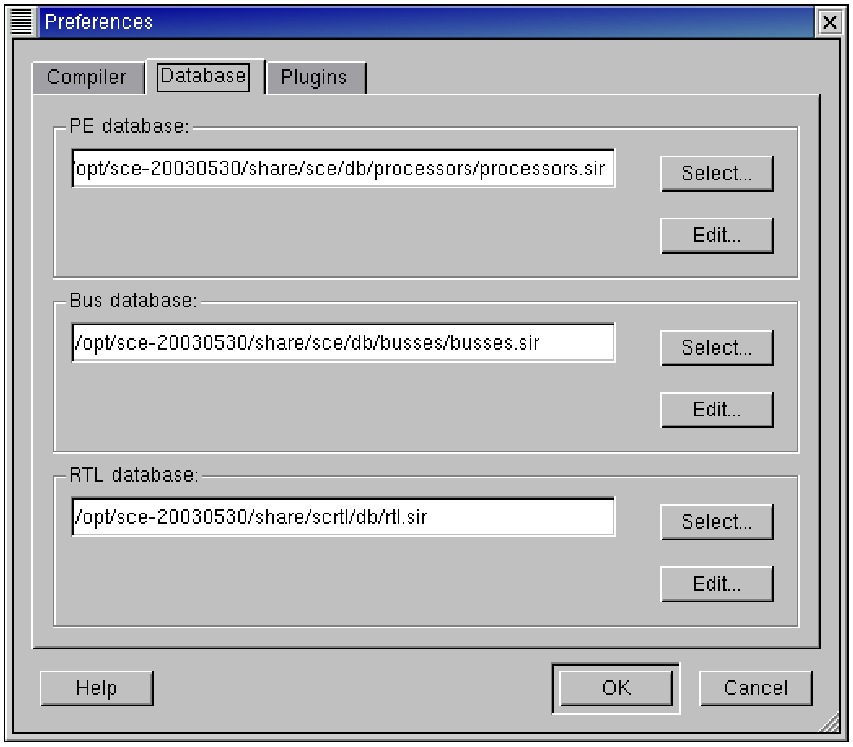
    Database Selection dialog allows users to choose and select existing database files on disk to use for each of the three databases.

    In the Database Selection dialog, users should first specify the database directory in *Look-in* box. The content of the directory will be automatically displayed in the display box in the center. The database type in the *File type* box defaults to SIR files for databases but can be chosen by the user. All the database files with the specified type will be displayed in the display box. Users further type in the database name in *File name* box. Finally, by clicking *Open* button, the database with the specified name will be selected. If users click *Cancel* button, then the action of database selection will be cancelled. Either clicking *Open* or *Cancel* button will close the Database Selection dialog.

Buttons *Ok* and *Cancel* appear at the bottom of the Edit Preference dialog. If users click the *Ok* button, all the edited preferences are saved. If users click the *Cancel* button, all the edited preferences are discarded. Either clicking *Ok* or *Cancel* button will close Preference dialog.

4.2 Project Handling

Project handling deals with project issues. Project handling functionality is common and shared across all SCE tools. It allows for tracking of design meta-data over the whole lifetime of a design. A project acts as a unified container that holds all information related to a certain design at various levels of abstraction, i.e. it contains all the information that describes the organization of design files that are part of the project. Furthermore, a project contains project-specific settings that can override or extend application-specific settings (see Section 4.1) for compiler paths, options, etc. Specifically, a project contains the following information:

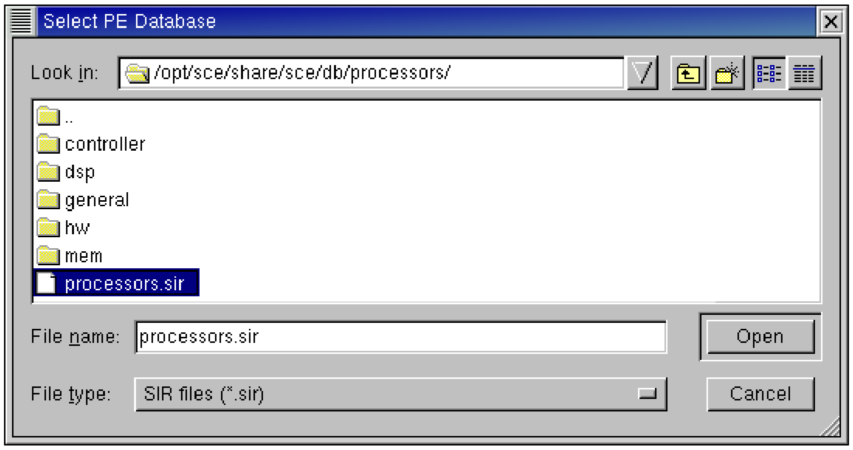


PE database: /opt/sce-20030530/share/sce/db/processors/processors.sir

Bus database: /opt/sce-20030530/share/sce/db/busses/busses.sir

RTL database: /opt/sce-20030530/share/scrtl/db/rtl.sir

Figure 15: Edit Preferences dialog (Database tab).



Look in: /opt/sce/share/sce/db/processors/

..
controller
dsp
general
hw
mem
processors.sir

File name: processors.sir

File type: SIR files (*.sir)

Figure 16: Database Selection dialog.

Design models A tree of design files and their relationship. If a design has been generated from another design through refinement, it is a child of the source design in the tree. For each model, the tree stores the design name, the location of the design's files on disk, the abstraction level, and the command used to generate the model.

Imports A list of imported design files. The list of imports contains the union of all (sub-)designs imported by any of the models that are part of the project.

Sources A list of source files. The list of sources contains the union of all SpecC source files from which the models that are part of the project have been compiled. For each source file, the location (path) of the file on disk is stored in the project.

Compiler settings A set of project-specific options for preprocessing and parsing SpecC source files. Compiler settings contain include paths, import paths, compiler options, and macro defines and undefines. Project-specific compiler settings generally overwrite or extend the corresponding application-specific settings. In the case of paths, project paths are prepended to the standard paths defined in the application settings (i.e. they are prepended to the directory search list). In all other cases, options or macro defines/undefined are appended to the compiler command line after the standard options and macros defined in the application settings.

All paths in the project settings are defined to be relative to the location of the project file, i.e. relative paths in a project file are converted into absolute paths by appending the project file's directory during loading/opening of a project file. During saving/writing of project files, absolute paths are in turn converted back to relative paths if they point to a location below the target project file directory.

Projects are stored as text files on disk in a custom XML format. The project file format is the same for all tools in the SCE environment, i.e. a project file can be read, modified and written by any SCE tool.

Projects can be read from and saved as project files at any time in the SCE application. At any time, however, at maximum only one project can be open and loaded. While a certain project is open and loaded, its settings apply to all actions performed during that time. In addition, certain actions will automatically update and add data in the currently opened and loaded project.

In order to deal with management of projects, SCE supports a set of project handling functions. Specifically, project handling consists of the following functions:

(a) Project Creation to create a new projects (see Section 4.2.1).

(b) Project Opening to open and load existing projects from project files on disk (see Section 4.2.2).

(c) Project Saving to save the current project into a project file (see Section 4.2.3).

(d) Project Settings to edit the settings of the opened project (see Section 4.2.4).

(e) Design Adding to adds new design files into the opened project (see Section 4.2.5).

(f) Design Opening to open a design model or import that is part of the project (see Section 4.2.6).

(g) Design Deletion to delete a design model file from the project and optionally the disk (see Section 4.2.7).

(h) Design Renaming to rename a design model and design file in the project and on disk (see Section 4.2.8).

(i) Description Changing to change the description of a design model in the opened project (see Section 4.2.9).

(j) Project Closing to close the current project (see Section 4.2.10).

4.2.1 Project Creation

Operation Users can create a new project by selecting $Main :: Project \Rightarrow New$.

Error/Information Messages Assuming before project creation, users have opened another project in SCE, the currently opened project has been modified and the opened project is not saved yet. When users select $Main :: Project \Rightarrow New$, an Information dialog will be popped up querying the user whether he wants to save the current project first before creating a new one. If the user accepts the recommendation, a Project Saving action (see Section 4.2.3) is performed first.

4.2.2 Project Opening

Operation Users open an existing project by selecting $Main :: Project \Rightarrow Open$. The selection will pop-up the Project Open dialog in which the user can choose and select an existing project file on disk to open and load. The screenshot of Project Open dialog is shown in Figure 17.

Users should first specify the project directory in *Look-in* box. The content of the directory will be automatically displayed in the display box in the center. The file type defaults to project files (with a '.sce' suffix) but users can specify any file type in the *File type* box. All the project files with the specified type will be displayed in the display box. Users further select the project name in the *File name* box. Finally, by clicking *Open* button, the project with the specified name will be opened. If users click the *Cancel* button, the action of project opening will be cancelled. Either clicking *Open* or *Cancel* button will close the Project Open dialog.

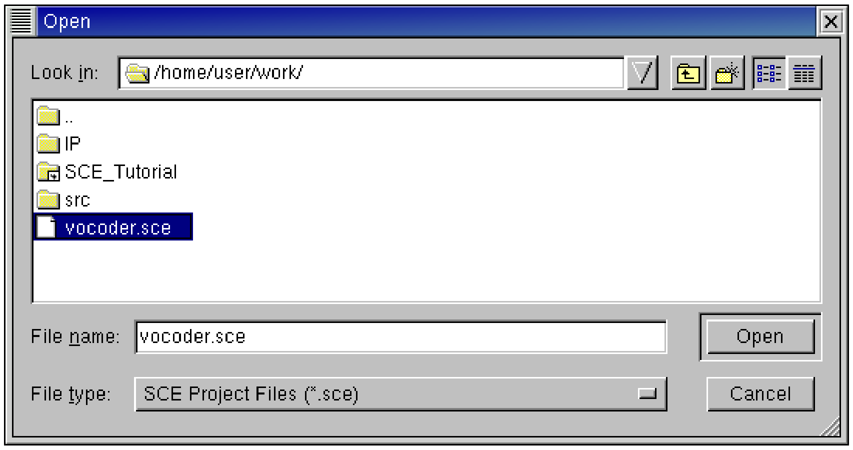


Figure 17: Project Open dialog.

Error/Information Messages If the specified project doesn't exist before clicking *Open* button, then clicking *Open* button has no effect.

In case of errors reading the project file from disk (file errors, wrong file format), an error dialog with a corresponding error message is popped up. Upon confirming the error, the Project Opening action is cancelled.

Assuming before project opening, users have opened another project in SCE, the opened project is modified and the opened project is not saved yet. When users open a different project, the Information dialog will be popped up to recommend users to save the previous project first and, if the recommendation is accepted, a Project Saving action will be performed. This is the same as the case in task Figure 4.2.1.

4.2.3 Project Saving

Operation Users can save the current project by one of the following two methods:

(a) Selecting *Main* :: *Project* ⇒ *Save*. The project will be saved using the current project name.

If the saved project is unnamed (a new project created by task Project Creating), then selecting *Main* :: *Project* ⇒ *Save* will do the same action as selecting *Main* :: *Project* ⇒ *Save As* (see below).

(b) Users can save the current project under any (new) name by selecting *Main* :: *Project* ⇒ *Save As*. The selection will pop-up the Project Save dialog in which user can choose the directory and file name to save the project under. The screenshot of the Project Save dialog is displayed in Figure 18.

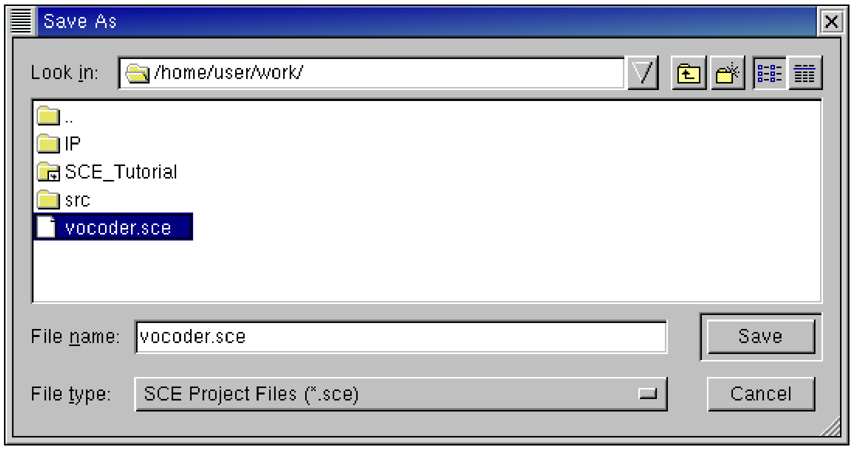


Figure 18: Project Save dialog.

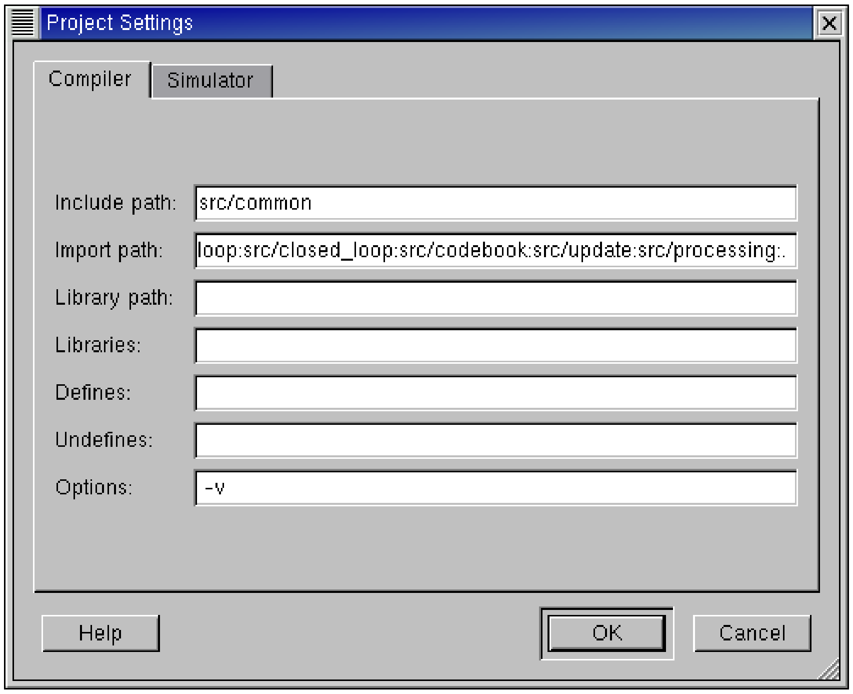
In the Project Save dialog, users should first specify the project directory in *Look-in* box. The content of the directory will be automatically displayed in the display box in the center. The file type defaults to project files ('.sce' suffix) but users can specify any file type in the *File type* box. All the project files with the specified type will be displayed in the display box. Users then select the project name in *File name* box. Finally, by clicking the *Save* button, the current project will be saved in a project file with the specified name. If users click *Cancel* button, then the action of project saving will be cancelled. Either clicking *Save* or *Cancel* button will close the Project Save dialog.

Error/Information Messages When selecting $Main :: Project \Rightarrow Save\ As$ and specifying the file name of an existing file on disk, an Information dialog will pop up asking the user whether he wants to overwrite the existing file. If the user declines this, the Project Saving action will be cancelled.

When selecting $Main :: Project \Rightarrow Save$ or $Main :: Project \Rightarrow Save\ As$, errors may occur (file errors, e.g. if no space is available on the hard disk). In this case, an Error dialog as shown in Figure 12 will be popped up, corresponding error messages will be displayed, and the Project Saving action will be cancelled.

4.2.4 Project Settings Editing

Operation Project setting allows users to edit project settings. Unlike application preferences editing in Section 4.1.1, project setting apply only to the current project. Users start project settings editing by selecting $Main :: Project \Rightarrow Setting$. The selection will pop up the Project Settings dialog, which is displayed in Figure 19. In the Project Settings dialog, user can view and edit the compiler and simulator settings stored in the project. For exmample, The dialog for compiler setting contains line edit boxes for all compiler settings. The text in the *Include path* and *Import path* lines defines the directory lists (separated by colons) for the project-specific include and import paths, respectively. The text in the *Defines* and *Undefines* lines define the list of macro defines and undefines (separated by semicolons), respectively. Finally, the text in the *Options* line defines the project's compiler options/switches.



Include path: src/common
Import path: loop:src/closed_loop:src/codebook:src/update:src/processing:.
Library path:
Libraries:
Defines:
Undefines:
Options: -v

Figure 19: Project Settings dialog.

4.2.5 Design Adding

Operation Users can add any currently opened design file to the project. In order to do that, users select *Main* :: *Project* ⇒ *Add Design*. After clicking, the design corresponding to the currently active design window will be added to the project. The design will be added to the project as a new root in the forest of design model trees. In addition, any imported designs and source files of the new design will be merged into the list of imports and sources that are part of the project.

Error/Information Messages If users try to add a file which is already in the current project, an error dialog with a corresponding error message will be popped up (as illustrated in Figure 12).

4.2.6 Design Opening

Operation Double-clicking on a design model in the Project Window models or import tabs ($Project :: Models$ or $Project :: Imports$), a File Opening action on the given design file will be performed (see Section 4.3.1 for details, including error/information messages), i.e. a corresponding design window will be opened in the workspace. If the selected design is already opened in the workspace, its design window will be raised to the top and made active. The Design Opening action can also be triggered via corresponding context menu entries in the model and import tabs (selecting $Project :: Models \Rightarrow Open$ or $Project :: Imports \Rightarrow Open$).

4.2.7 Design Deletion

Operation Users can delete files from the current project and optionally from disk. Selecting the corresponding entry in the context menu of a design model in the Project Window models tab ($Project :: Models \Rightarrow Delete$) will delete the selected design from the project and optionally from disk. In case of a model with children in the model tree, the user will also be given the option to recursively delete all the model's children.

Error/Information Messages After selecting $Project :: Models \Rightarrow Delete$, an Information dialog will be popped up to query to user whether he wants to also delete the corresponding model's files on disk. If the selected file has children, then another Information dialog querying the user whether he wants to also recursively delete all children and children's children of the selected model. If the selects recursive deletion, an Information dialog similar to the initial Information dialog to query about deletion of corresponding files on disk will pop up for each child model.

If file deletion on disk is selected, an error dialog may pop up in case of disk/file errors. Upon confirmation of the error, the Project Deletion action will be aborted.

4.2.8 Design Renaming

Operation By selecting $Project :: Models \Rightarrow Rename$ or by clicking into the *Design* column of the Project Window models tab, users can rename the file name displayed in the *Design* column. Renaming is performed in place inside the column cell itself by opening a corresponding text edit box. Renaming can be aborted by pressing the *Esc* key. Pressing *Enter* accepts the newly entered name and renames the design both in the project and on disk.

If the design is loaded and opened, the corresponding design window in the Workspace will also be automatically renamed.

Error/Information Messages If the new design name entered by the user is the name of a design already existing in the project, an Error dialog with a corresponding error message will pop up and, after confirmation, the Design Renaming action will be aborted.

If renaming the file on disk results in an error (file error), a corresponding Error dialog will be popped up and the Design Renaming will be aborted.

4.2.9 Description Changing

Operation Users can change the file description displayed in the *Description* column of $Project :: Models$ tab by selecting the $Project :: Models \Rightarrow Change\ Description$ context menu entry or by clicking into the corresponding *Description* column in the row representing the selected file. After clicking, the corresponding cell in the *Description* column of $Project :: Models$ is editable in place. Editing can be aborted with *Esc* and is accepted by pressing *Enter*.

4.2.10 Project Closing

Operation Users can close the current project by selecting $Main :: Project \Rightarrow Close$.

Error/Information Messages If the current project is modified and not yet saved, selecting *Close* will pop up an Information dialog which recommends to save the current project first. If the user accepts the recommendation, a Project Saving action (Section 4.2.3) is performed before closing the project.

4.3 File Handling

File handling deals with issues relating to manipulation of design files within SCE. File handling includes opening, saving, and closing of actual design model files on disk. File handling is closely related to Design Windows (Section 3.3) and Design Window Management (Section 4.6). In general, there is a one-to-one association between design models, design files on disk and design windows in the Workspace. Each Design Window represents a view onto one loaded design file which in turn stores the data of one design model, and vice versa. For example, both File Closing (Section 4.3.3) and Window Closing (Section 4.6) will close the design file and the design window and unload the design from SCE's memory. Specifically, File Handling consists of the following tasks:

(a) File Opening to open and load existing design files from disk (see Section 4.3.1).

(b) File Saving to save a design into a design file on disk (see Section 4.3.2).

(c) File Closing to close a design file (see Section 4.3.3).

(d) File Import to import an existing design file from disk into a currently opened design (see Section 4.3.4).

(e) Design Property to display the current design file's properties (see Section 4.3.5).

(f) SCE Exiting to exit the SCE application (see Section 4.3.6).

4.3.1 File Opening

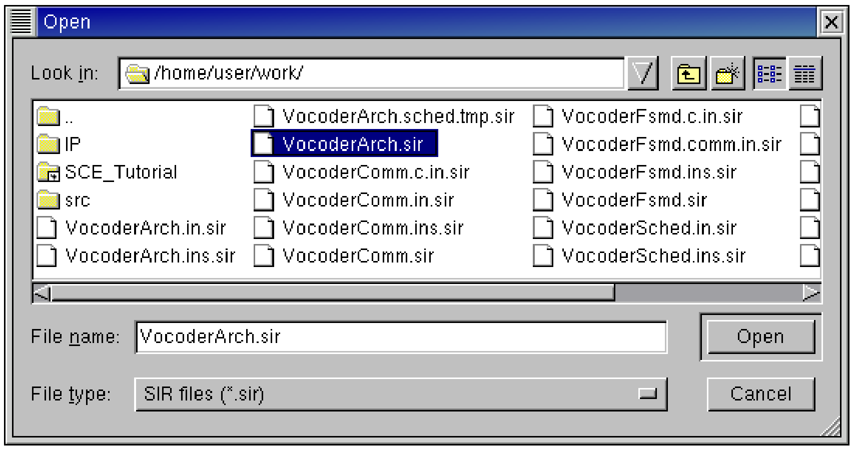


Figure 20: File Open dialog.

Operation Users can open an existing design file on disk in different ways:

(a) Selecting *Main* :: *File* ⇒ *Open* will pop up the File Open dialog in which the user can choose and select an existing file on disk to open and load. The File Open dialog is illustrated in Figure 20.

Users should first specify the directory of the file in *Look-in* box. The content of the directory will be automatically displayed in the display box in the center. The file type defaults to SpecC source files ('.sc' suffix) but user can specify any file type in the *File type* box. All the files with the specified type will be displayed in the display box. Users then further select the file name in the *File name* box. Finally, by clicking the *Open* button, the file with the specified name will be open. If users click the *Cancel* button, the action of file opening will be cancelled. Either clicking *Open* or *Cancel* button will close the File Open dialog.

(b) Double-clicking on a design in the Project Window models or imports tabs (*Project* :: *Models* or *Project* :: *Imports*) or selecting the corresponding context-menu entries (*Project* :: *Models* ⇒ *Open* or *Project* :: *Imports* ⇒ *Open*) will open and load the design file for the selected design model.

Opening and loading a design file in either way will result in a corresponding new Design Window popping up in the Workspace. The new Design Window will automatically be made the active one and raised to the front of the Workspace.

If the selected design file is already opened and loaded, it will not be loaded again from disk and File Opening will only result in activating and raising the corresponding Design Window in the Workspace.

Error/Information Messages In the case of selecting from disk, if the specified file doesn't exist before clicking the *Open* button, then clicking *Open* has no effect.

In case of errors reading the design file from disk (file errors, wrong file format), an error dialog with a corresponding error message is popped up. Upon confirming the error, the File Opening action is cancelled.

4.3.2 File Saving

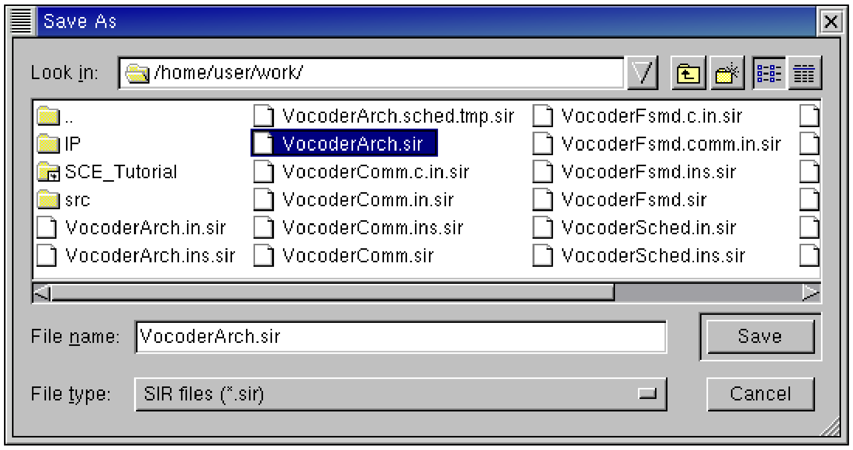


Figure 21: File Save dialog.

Operation Users can save opened and loaded design files (Design Windows in the Workspace) by one of the following three methods:

(a) Selecting *Main* :: *File* ⇒ *Save* will save the file of the currently active Design Window using its current name.

(b) Users can save the file of the currently active Design Window under any (new) name by selecting *Main* :: *File* ⇒ *Save As*. The selection will pop up the File Save dialog in which user can choose the directory and file name to save the design under. The screenshot of the File Save dialog is shown in Figure 21. The solid rectangular represents the edit box.

    In the File Save dialog, users should first specify the directory of the file in *Look-in* box. The content of directory will be automatically displayed in the display box in the center. The file type defaults to SpecC source files ('.sc' suffix) but users can specify any file type in the *File type* box. All the files with the specified type will be displayed in the display box. Users then further select the file name in *File name* box. Finally, by clicking *Save* button, the current opened file will be saved as the file with the specified name. If users click *Cancel* button, then the action of File Saving will be cancelled. Either clicking *Save* or *Cancel* button will close the File Save dialog.

(c) Selecting *Main* :: *File* ⇒ *Save All* will save the files of all currently opened Design Windows in the Workspace using their current names.

Error/Recommendation Messages. When selecting *Main* :: *File* ⇒ *Save As* and specifying the file name of an existing file on disk, an Information dialog will pop up asking the user whether he wants to overwrite the existing file. If the user declines this, the File Saving action will be cancelled.

When selecting *Main* :: *File* ⇒ *Save*, *Main* :: *File* ⇒ *Save As*, or *Main* :: *File* ⇒ *Save All*, errors may occur (file errors, e.g. if no space is available on the disk). In this case, an Error dialog as shown in Figure 12 will be popped up, corresponding error messages will be displayed, and the File Saving action will be cancelled.

4.3.3 File Closing

Operation Users can close the file and window of the currently active Design Window in the Workspace by selecting *Main* :: *File* ⇒ *Close*. Closing a file will unload the design from memory and will close the corresponding Design Window in the Workspace.

Error/Information Messages If the current design is modified and not yet saved, selecting *Close* will pop up an Information dialog which recommends to save the current design first. If the user accepts the recommendation, a File Saving action (Section 4.3.2) is performed before closing the file.

4.3.4 File Import

Operation Users can import an existing design file on disk into a currently opened and loaded design. Importing a design will merge the design's contents into the currently opened design (equivalent to a SpecC `import` statement). All the design entities in the imported files can then be used in the current design. For example, in order to do plug-and-play of behaviors, users can replace a behavior in a design with an imported compatible behavior (see Section 4.4.2 for the Changing Type action).

In order to import a file users should first active the target Design Window in the Workspace and then select *Main* :: *File* ⇒ *Import*. The selection will pop up File Importing dialog in which users can select and choose an existing design file on disk to open and import into the currently active Design Window. The File Importing dialog is shown in Figure 22.

In the File Importing dialog, users should first specify the directory of the file in *Look-in* box. The content of the directory will be automatically displayed in the display box in the center. The file type defaults to SpecC source files ('.sc' suffix) but users can specify any file type in the *File type* box. All the files with the specified type will be displayed in the display box. Users then further select the file name in the *File name* box. Finally, by clicking the *Open* button, the file with the specified name will be imported. If users click *Cancel* button, then the action of File Importing will be cancelled. Either clicking *Open* or *Cancel* button will close the File Import dialog.

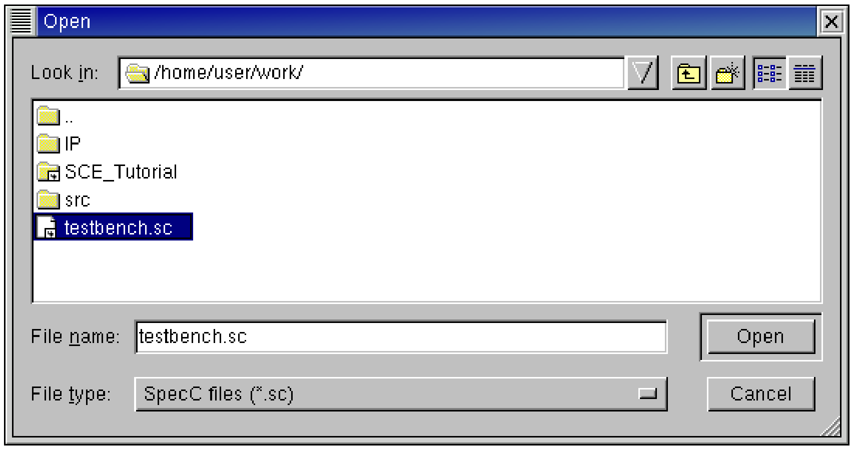
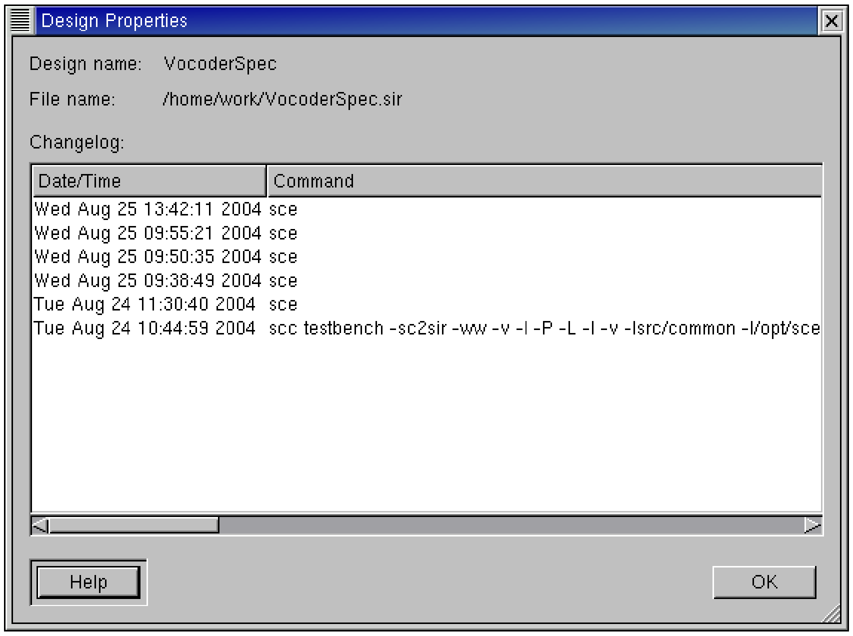


Figure 22: File Importing dialog.

4.3.5 Design Property Viewing



Design name: VocoderSpec
File name: /home/work/VocoderSpec.sir
Changelog:

Date/Time	Command
Wed Aug 25 13:42:11 2004	sce
Wed Aug 25 09:55:21 2004	sce
Wed Aug 25 09:50:35 2004	sce
Wed Aug 25 09:38:49 2004	sce
Tue Aug 24 11:30:40 2004	sce
Tue Aug 24 10:44:59 2004	scc testbench -sc2sir -ww -v -I -P -L -I -v -Isrc/common -I/opt/sce

Figure 23: Design Property dialog.

Operation Users can display and view a design's properties by selecting *Main* :: *File* ⇒ *Properties*. The selection will pop up a Design Properties dialog which is illustrated in Figure 23.

In the Design Property dialog, the box *Design Name* displays the design name (*VocoderSpec* in Figure 23), the Box *File name* displays the name of the design file (e.g. */home/work/VocoderSpec.sir*), and the box *Changelog* displays the history of SCE refinement commands applied to the design. The left part of the *Changelog* specifies the Date/Time of the executed command. The right part of *Changelog* specifies the exact command lines with parameters. Clicking button *Ok* will close Design Properties dialog .

4.3.6 SCE Exiting

Operation Selecting *Main* :: *File* ⇒ *Exit* will exit the SCE application and close the SCE GUI.

The current project, if any, will be automatically saved by triggering a Project Saving action (see Section 4.2.3). Note that this can result in a Project Save dialog popping up in case the current project is unnamed and modified.

Error/Recommendation Messages. If there are any open Design Windows that are modified and not yet saved, an Information dialog will pop up for each window, querying the user whether he wants to save the corresponding design. The user will be able to cancel the whole Exit action via the corresponding dialog button. If the user accepts the recommendation to save the file, a File Saving action will be triggered (see Section 4.3.2). Note that the File Saving action can trigger additional Error dialogs which in turn canabort the whole Exit operation in case of file errors during saving.

Automatic Project Saving can result in errors and corresponding Error dialogs popping up. In turn, the whole Exit operation can be aborted in those cases.

4.4 Design-Entity Handling

Design-Entity Handling deals with manipulation of design entities such as behaviors, channels, variables, and channels. In general, Design entities are manipulated directly in the currently active Design Window in the Workspace. Design-Entity Handling is part of the refinement process and allows the user to perform some typical refinement tasks in a manual fashion. Basically, Design-Entity Handling tasks are tasks that do not change the semantics of the design. Rather, they are tasks to apply, for example, cosmetic or syntactic changes like behavior renaming to the design. Specifically, Design-Entity Handling consists of the following tasks:

(a) Entity Renaming to rename the selected entity (see Section 4.4.1).

(b) Entity Retyping to change the type of the selected entity (see Section 4.4.2).

(c) Entity Deletion to delete the selected entity from the design (see Section 4.4.3).

(d) Hierarchy Displaying to toggle between different modes for displaying of the design hierarchy (see Section 4.4.4).

4.4.1 Entity Renaming

Operation The names of design entities including behaviors, behavior instances, channels, channel instances, variables, and ports are displayed in *Name* column of the Design Window tabs. Users can change these names of entities. In order to do it, users first select the corresponding row in *Design* :: *Hierarchy*, *Design* :: *Behaviors* or *Design* :: *Channels* tabs. Selecting *Design* :: *Hierarchy* ⇒ *Rename*, *Design* :: *Behaviors* ⇒ *Rename*, or *Design* :: *Channels* ⇒ *Rename* context-menu commands will then allow users to edit the names directly in the cell by opening a corresponding text edit box in place. Renaming can be aborted by pressing the *Esc* key. Pressing *Enter* accepts the newly entered name and renames the entity in the design.

Error/Information Messages If renaming the entity in the design leads to an error (e.g. if an entity with the same name already exists), an Error dialog with a corresponding error message will pop up and subsequently the operation will be aborted.

4.4.2 Entity Retyping

Operation Users can change the type of design entity instances including variable, channel and behavior instances that are displayed in the *Type* column of Design Window hierarchy tab. In order to do it, users first need to select the corresponding row in the *Design* :: *Hierarchy* tab. Selecting the *Design* :: *Hierarchy* ⇒ *Change Type* context-menu command will then allow users to chose a type for the instance entity from a list of compatible types directly in the cell by opening a drop-down combo box with entries for all compatible types in place. Compatibility in SpecC is defined by a match in the list of port types and the list of implemented interfaces, e.g. two behaviors are compatible if they have the same list of ports in terms of order and type of ports. Users can choose any of the offered types from the drop-down list. Selecting a new type will then change the type of the instance entity in the design.

Error/Information Messages If retyping of the entity in the design leads to an error, an Error dialog with a corresponding error message will pop up and subsequently the operation will be aborted.

4.4.3 Entity Deletion

Operation Users can delete unused design entities from the design. In order to do it, users first need to select the corresponding row in *Design* :: *Hierarchy*, *Design* :: *Behaviors*, or *Design* :: *Channels* tabs. Selecting *Design* :: *Hierarchy* ⇒ *Delete*, *Design* :: *Behaviors* ⇒ *Delete* or *Design* :: *Channels* ⇒ *Delete* context-menu commands will then delete the selected entity from the design.

Error/Information Messages If deleting the entity in the design leads to an error (e.g. if the entity is used by another entity, i.e. if some part of the design depends on that entity), an Error dialog with a corresponding error message will pop up and subsequently the operation will be aborted.

4.4.4 Hierarchy Displaying

Operation Users can toggle between different modes for displaying the design hierarchy in the Design Window hierarchy tab. Selecting *Main* :: *View* ⇒ *Hierarchy* will open a sub-menu pop-up containing for entries corresponding to the four different display modes:

Show All Displays a forest of instantiation hierarchy trees where all uninstantiated behaviors are roots and the complete instantiation tree for all of them is shown.

Show Testbench Displays the complete instantiation hierarchy tree for the *Main* behavior class root only.

Show Design Displays the instantiation hierarchy tree only for the root class that is currently the top-level design behavior. This basically shows the current system being designed.

Show Architecture Displays the current top-level behavior as the only root class and only displays first-order (direct) children of (instances in) that behavior. This basically shows only the top, system level of the current system being designed.

Display modes are mutually exclusive, i.e. selecting one display mode will turn off the previous mode and switch to the new mode instead.

In addition to switching hierarchy display modes, users can toggle displaying of both variables and channels in the hierarchy tree. Selecting *Main* :: *View* :: *Show Variables* or *Main* :: *View* :: *Show Channels* toggles between display of variables and channels in the hierarchy tab of the current Design Window. If channels are displayed, both channel instances and uninstantiated channels are shown in the forest of instantiation hierarchy trees. If variables are displayed, both global variables and variable instances inside behaviors (and optionally channels) are shown in the forest of instantiation hierarchy trees.

4.5 Synthesis

4.5.1 Architecture Synthesis

Architecture Synthesis deals with the process of implementing a specification on a computation architecture consisting of PEs and memories in order to generate a respective architecture model for the design. Architecture synthesis therefore helps designers to allocate PEs/memories, map design entities to the allocated PEs/memories and to generate the Architecture Model. Specifically, Architecture Synthesis consists of four tasks:

(a) Top-Level Selection to select the behavior representing the top level of the system to be designed (see Section 4.5.1).

(b) PE Allocation to allocate and select PEs/memories from the PE database in order to assemble the system architecture (see Section 4.5.1).

(c) Mapping to map the design's computation entities to the selected PEs/memories (see Section 4.5.1).

(d) Architecture Refinement to automatically generate an Architecture Model from the given Specification Model based on the decision made during PE Allocation and Mapping (see Section 4.5.1).

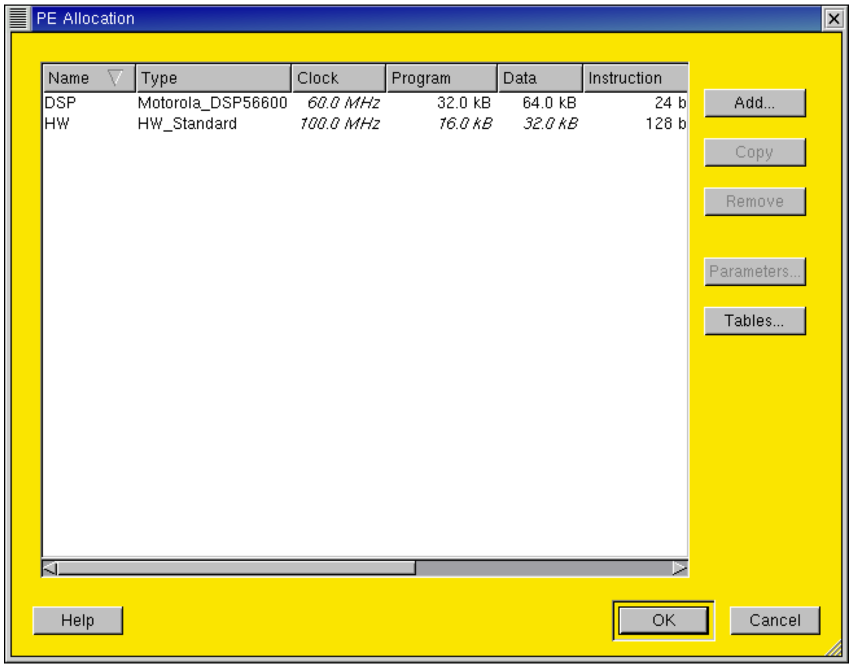
Top-Level Selection Users can select the top-level behavior which represents the system design to be implemented. All the children of the top-level behavior are considered to be part of the system design. On the other hand, All the parents/siblings of the top-level behavior are considered to belong to the testbench outside of the actual design.

Operation Users first select the desired behavior by clicking its row in the $Design :: Hierarchy$ or $Design :: Behaviors$ tabs. Then, users set the selected behavior as the top-level behavior by selecting the $Design :: Hierarchy \Rightarrow Set\ As\ Top-Level$ or $Design :: Behaviors \Rightarrow Set\ As\ Top-Level$ commands in the context-menu for the respective behavior.

After selection, the instantiation hierarchy tree with the current top-level behavior at its root will be visually emphasized/highlighted in the Design Window's hierarchy tab (through use of an italic font for entity texts).

Error/Information Messages All allocation information is stored in the design as annotations at the top-level design behavior (see Section 4.5.1). When switching the top-level from one behavior to another, if an allocation exists at the current behavior, an Information dialog is popped up querying the user whether he wants to copy the existing allocation over to the new top-level behavior. If the user accepts the recommendation and if then an allocation already exists at the new top-level behavior, another Information dialog is popped up querying the user whether he wants to overwrite the allocation information at the new top-level behavior. If the user declines the recommendation, the previous allocation at the new top-level behavior is kept.

PE Allocation Users can select PEs/memories out of the PE database in order to allocate and assemble the system architecture. PE allocation information is stored in the design itself as an allocation table that is annotated at the top-level design behavior (see Section 4.5.1). As a consequence, different allocation tables at different top-level behaviors can exist in the same design, reflecting the fact that incremental design will require changes in the allocation as design progresses from one part of the system to another.



PE Allocation

Name	Type	Clock	Program	Data	Instruction
DSP	Motorola_DSP56600	*60.0 MHz*	32.0 kB	64.0 kB	24 b
HW	HW_Standard	*100.0 MHz*	*16.0 kB*	*32.0 kB*	128 b

Figure 24: PE Allocation dialog.

Operation In order to do PE Allocation, users first select *Main* :: *Synthesis* ⇒ *Allocate PEs*. As a result, the current allocation is read from the design and a PE Allocation dialog is popped up. In case of errors reading the allocation from the design (e.g. wrong allocation table format), a new, empty allocation table is used in the dialog.

The screenshot of the PE Allocation dialog is shown in Figure 24. In the PE Allocation dialog, the table shows the list of currently allocated PEs. The header of the table indicates the meaning of each column, such as PE's name and type. Each row in the table represents an allocated PE/memory that is part of the current system target architecture. For each PE, its name, its type, its attributes (ex. Clock, Program, Data, Instruction) are shown in the respective columns of the table. The list of allocated PEs can be sorted by any column and in ascending or descending order each by clicking into the corresponding column header. By default, the list is sorted by ascending names.

In the PE Allocation dialog, users can perform the following actions:

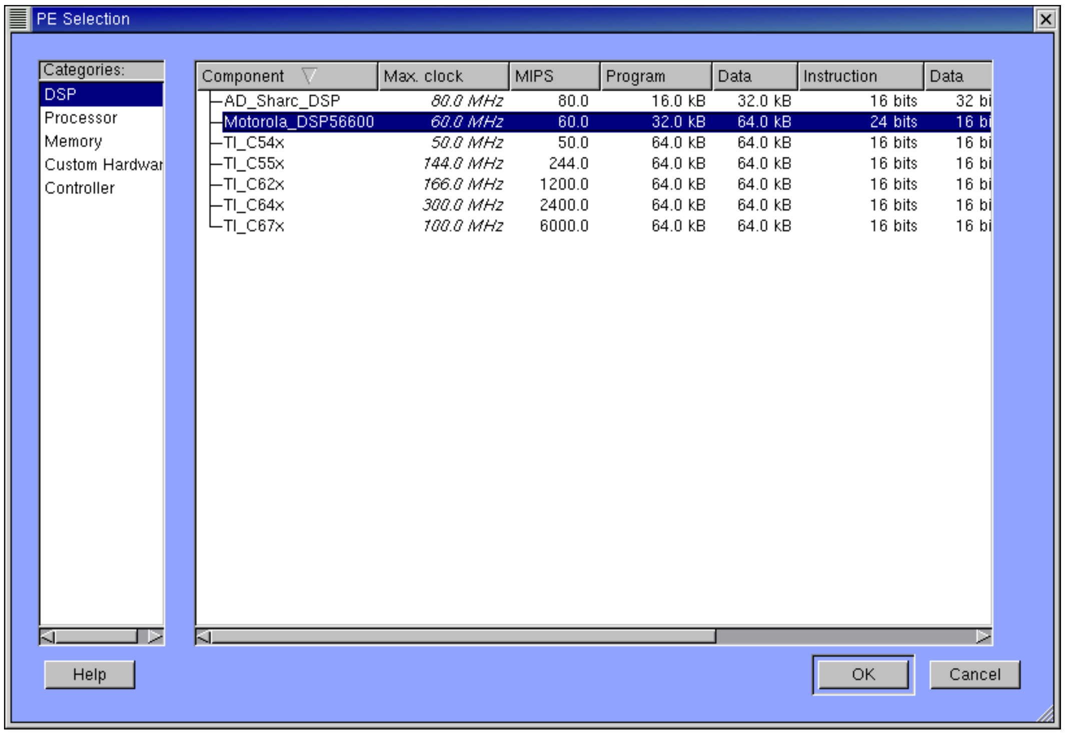
PE Adding In order to add a PE into the design, users click the button *Add* to pop up the PE Selection dialog which opens and loads the PE database and allows users to select an additional PE out of the PE database. The screenshot of the PE Selection dialog is shown in Figure 25.

At the left of the PE Selection dialog is a PE category table. Each row represents one category of PEs in the database. For example, row *Processor* contains all the general-purpose processors in the database.

By clicking and selecting one row in the table at left, users will be shown all the PEs in the selected category in the table at the right. Each row of the table at the right represents one type of PE in the database under the selected category. The name of the PE type (*Component* column) and other attributes of the PE type are displayed in separate columns. Users can select the desired PE type by clicking the corresponding row.

There are two buttons at the bottom of the PE Selection dialog: *Ok* and *Cancel*. By clicking the *Ok* button, an additional PE of the selected PE type and with an automatically determined name is added into the design's allocated architecture. In addition, PEs can be allocated by double-clicking into the desired PE type row in the PE Selection dialog (equivalent to selecting the row and pressing *Ok*). Clicking the *Cancel* button aborts and cancels PE selection without changes to the PE allocation. Either clicking *Ok* or *Cancel* button will close the PE Selection dialog and return to the PE Allocation dialog.

PE Copying In order to duplicate an existing PE in the design's PE allocation, users can select a PE by clicking the corresponding row in the allocation table and click the button *Copy*. Clicking *Copy* will add a new PE instance with an automatically de-



Component	Max. clock	MIPS	Program	Data	Instruction	Data
AD_Sharc_DSP	80.0 MHz	80.0	16.0 kB	32.0 kB	16 bits	32 bi
Motorola_DSP56600	60.0 MHz	60.0	32.0 kB	64.0 kB	24 bits	16 bi
TI_C54x	50.0 MHz	50.0	64.0 kB	64.0 kB	16 bits	16 bi
TI_C55x	144.0 MHz	244.0	64.0 kB	64.0 kB	16 bits	16 bi
TI_C62x	166.0 MHz	1200.0	64.0 kB	64.0 kB	16 bits	16 bi
TI_C64x	300.0 MHz	2400.0	64.0 kB	64.0 kB	16 bits	16 bi
TI_C67x	100.0 MHz	6000.0	64.0 kB	64.0 kB	16 bits	16 bi

Figure 25: PE Selection dialog.

termined name and with the same type, attributes and description as the currently selected PE to the design's allocation.

PE Deletion . In order to remove a PE from the design's allocation, users can select the target PE to be removed in the allocation table and click the *Remove* button. Clicking *Remove* will remove the selected PE from the list of allocated PEs.

PE Editing Users can edit the PE name and PE description in the PE Allocation dialog by clicking into the respective *Name* or *Description* column of the corresponding PE. Clicking into any of these cells will allow editing of the respective text in the cell by opening a text edit box in place. Pressing the *Esc* key during editing aborts the edit operation. Pressing *Enter* accepts the entered text and changes the PE name or description in the allocation accordingly.

Allocation Closing There are two buttons at the bottom of the PE Allocation dialog: *Ok* and *Cancel*. By clicking the *Ok* button, the allocation displayed in the PE Allocation dialog's allocation table is saved into the design. Clicking *Cancel* will abort and cancel PE Allocation. When cancelling, all modifications made to the allocation table will be lost and no data will be saved in the design. Either clicking *Ok* or *Cancel* button will close PE Allocation dialog.

Error/Information Messages There are several possible errors during PE Allocation:

(a) Before PE Allocation, selecting *Main* :: *Synthesis* ⇒ *Allocate PEs* if no top-level behavior is selected in the design will pop up an Error dialog to that effect and will abort the PE Allocation operation.

(b) During PE Editing, if users try to give PEs a name which is already used as the name of another PE in the design, an Error dialog will be popped up, a corresponding error message will be shown, and the editing operation will be aborted and cancelled.

(c) During PE Adding, when clicking the *Add* button the PE database stored on disk will be opened and loaded in order to read the list of available PEs from the database. In case of errors during database opening (e.g. file errors or wrong file format), an Error dialog will be popped up and the PE Adding operation will be aborted.

Furthermore, when adding a PE to the allocation via the PE Selection dialog, the selected PE type is read from the database. In case of database read errors (file errors, database format errors) during this operation, an Error dialog will be popped up and the PE Adding operation aborted.

(d) During PE Allocation, clicking the *Ok* button in the PE Allocation dialog will write the allocation table back to the design. In case of errors, an Error dialog is popped up and PE Allocation is aborted completely.

Mapping In order to implement the computation in the specification model on the allocated computation architecture consisting of PEs and memories, users have to be able to map the behaviors and variales in the specification onto the allocated PEs. Hence, Mapping consists of separate Behavior Mapping, Variable Mapping tasks:

Behavior Mapping Behavior Mapping allows for mapping of behavior types/classes in the design onto allocated PEs, i.e. behavior mapping information is stored as annotations at the behavior classes in the design. In order to be able to map a behavior onto a PE, the PE out of the database must allow execution of arbitrary code on it. Users can explicitly map every behavior type on a PE. Explicit mapping will map all instances of that behavior onto the selected PE. If instances should be mapped to different PEs, appropriate copies of the behavior have to be made outside of SCE first. If a behavior is not mapped to any PE, all of its instances will be implicitly (and recursively) mapped onto the same PE as the parent behavior class in which they are instantiated in. If different instances are implicitly mapped to different PEs, appropriate copies of the behavior in each PE will be automatically generated during refinement. Note that user must map all the behaviors under the top-level behavior to PEs either implicitly (by mapping the top-level behavior itself) or explicitly.

Variable Mapping Variable Mapping allows for mapping of variable instances in the design into local memories of allocated regular PEs or into allocated global, shared memory PEs. Variable mapping information is stored as annotations attached to variable definitions inside a behavior class. As a result, multiple incarnations of the same variable in different instances of the parent behavior will all share the same mapping information. If a variable is not explicitly mapped by the user, during refinement a local copy of the variable will be created in each PE accessing the variable. Refinement will also automatically insert necessary code (additional behaviors inside PEs and channels between PEs) for synchronization and message passing to keep copies updated and synchronized such that shared semantics are preserved. Implicit mapping is not supported for variables that are shared among concurrent behaviors mapped to different PEs. If a variable is explicitly mapped into local memory of a regular PE or into a shared, global memory PE, all of its incarnations will be moved there and other PEs will access the variable through a memory interface. Explicit mapping of variables is only supported for target PEs out of the database that support external accesses via a memory interface.

Operation Users can map behaviors, variables in the design to allocated regular or memory PEs via the additional *PE* column in the Design Window hierarchy tab. Note that the *PE* column is only shown if PE allocation information is available (see Section 4.5.1): By default, the *PE* column shows the current mapping information for each entity in the design. In case of errors reading the mapping information from the design (e.g. wrong annotation format), an empty, implicit (i.e. lack of explicit) mapping will be assumed.

In order to explicitly map an entity, users should click into the *PE* column of the respective entity in the *Design* :: *Hierarchy* tab. If the desired entities are not shown in the hierarchy tab, users should first enable display of variables or channels by selecting *Main* : *Synthesis* ⇒ *Show Variables* or *Main* : *Synthesis* ⇒ *Show Channel*.

Clicking into the *PE* column of the Design Window hierarchy tab will open a drop-down combo box in place with entries for all possible target PEs in order allow users to select a target PE to map the entity to directly in the cell. In the combo box, users will be able to choose from all possible PEs that the specific entity can be mapped to (see above for enforced restrictions). In addition, the combo box contains an empty entry to chose in order to remove any existing explicit mapping and switch to implicit mapping for that entity.

Selecting an entry from the combo box will write the corresponding mapping into the design. If there are multiple incarnations of an explicitly mapped entity (behavior class, variable definition or channel instance), the hierarchy tab display will be updated after changing the mapping to reflect the new mapping for all entity's incarnations in the *PE* column.

After mapping, the behaviors mapped to different PEs will be shown with different colors.

Architecture Refinement Architecture Refinement executes the implementation decisions made in the other Architecture Synthesis tasks by refining the current Specification Model into an automatically generated Architecture Model based on and reflecting the decision made during PE Allocation and Mapping.

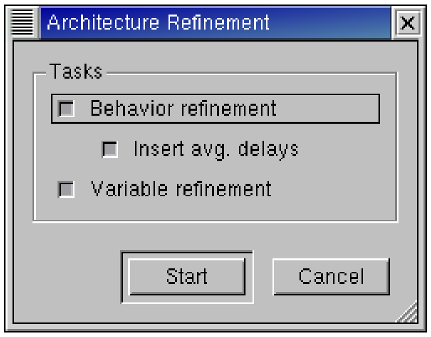


Figure 26: Architecture Refinement dialog.

Operation In order to do refinement, users can select the *Main* :: *Synthesis* ⇒ *Architecture Refinement* menu entry to pop up the Architecture Refinement dialog. The screenshot of the Architecture Refinement dialog is shown in Figure 26.

In the Architecture Refinement dialog, users can select whether individual sub-tasks of the architecture refinement process will be performed or not. By checking or unchecking the check boxes tasks are turned on and off and partially or completely refined models can be generated. By default, all tasks are turned on. The three sub-tasks of architecture refinement are:

(a) *Behavior Refinement* which introduces PE behaviors from the database, groups the original behaviors under the new PEs, and inserts synchronization and message passing to preserve execution semantics.

(b) *Insert avg. delays* which inserts the average delays generated by profiler to the system behavior.

(c) *Variable Refinement* which (re-)distributes variables into PEs, generates necessary PE memory interfaces, and updates accesses to shared variables inside leaf behaviors.

If *Behavior Refinement* is turned off, both *Insert avg. delays* and *Variable Refinement* are turned off and can not be turned on. In this case, no output model is generated and only input validation is performed. In other cases, exactly one output model is generated in which optionally variables and channels are not refined but left untouched.

User can then start the architecture refinement process by clicking the *Start* button. If users click the *Cancel* button, the Architecture Refinement operation will be aborted and cancelled. Either clicking *Start* or *Cancel* buttons will close the Architecture Refinement dialog.

After clicking the *Start* button, the architecture refinement command line components will be executed in the background. Any diagnostic, status and informative output of the architecture refinement tools will be shown in the Refinement tab of the Output Window ($Output :: Refine$).

When the architecture refinement process is finished, the newly generated architecture model is automatically opened and loaded, and a corresponding new Design Window is created in the Workspace. The new Design Window is automatically activated and raised to the front. In addition, the new architecture model is automatically added to the current project (see Design Adding, Section 4.2.5) as a child of the specification model it was generated from.

While the architecture refinement background tools are running, the majority of the main SCE GUI is disabled. However, users can abort/kill execution of the background tools by selecting $Main :: Synthesis \Rightarrow Stop$. After clicking, the current architecture refinement background task is aborted.

Error/Information Messages If the architecture refinement background tools abort with an error (e.g. unmapped behaviors in the design) or are killed via the $Main$ ::

Synthesis ⇒ *Stop* menu entry, an Error dialog with a corresponding error message will pop up. Specifically, the architecture refinement background tools check for and can produce the following classes of errors:

- No top level behavior.
- No or invalid PE allocation.
- No or invalid PE models in the database.
- Unsupported (partitioned) behavior types in the specification.
- Unsupported (global or partitioned) shared variable and/or port types in the specification.
- Invalid variable mapping or no mapping for variables shared between concurrent behaviors on different PEs.

Upon confirming the error, the remainder of the Architecture Refinement Operation will be cancelled.

If the design the new model was generated from is not in the project, the new model is not added to the project and an Error dialog to that effect will be popped up.

4.5.2 Communication Synthesis

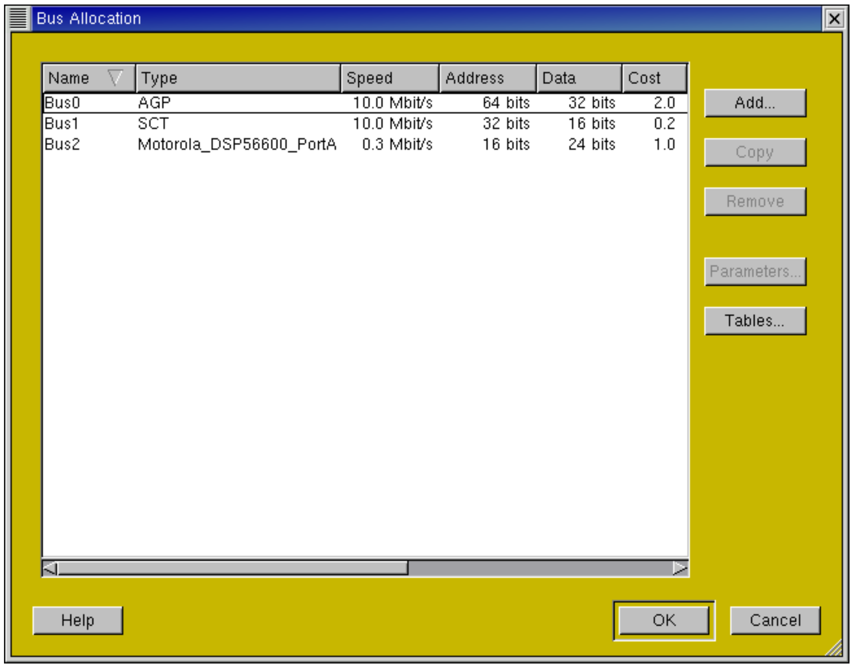
Communication Synthesis refines the abstract communication between components in the architecture model into an actual implementation over wires and protocols of system buses. Specifically, Communication Synthesis consists of three tasks:

(a) Bus Allocation to allocate buses.

(b) Channel Mapping to map channels to the selected buses.

(c) Communication Refinement to automatically generate a Communication model from the given Architecture Model based on the decisions made during Bus Allocation and Channel Mapping.

Bus Allocation In order to define the system network topology, users can allocate buses out of the bus databases. Bus allocation is stored in the design itself as allocation that are annotated at the top-level design behavior (see Section 4.5.1). As a consequence, different allocation tables at different top-level behaviors can exist in the same design, reflecting the fact that incremental design will require changes in the topology as design progresses from one part of the system to another.



Name	Type	Speed	Address	Data	Cost
Bus0	AGP	10.0 Mbit/s	64 bits	32 bits	2.0
Bus1	SCT	10.0 Mbit/s	32 bits	16 bits	0.2
Bus2	Motorola_DSP56600_PortA	0.3 Mbit/s	16 bits	24 bits	1.0

Figure 27: Bus Allocation dialog.

Operation In order to do Bus Allocation, users first select *Main* :: *Synthesis* ⇒ *Allocate Buses*. As a result, the current allocation is read from the design and a Bus Allocation dialog is popped up. In case of errors reading the allocation from the design (e.g. wrong allocation table format), new, empty allocation tables are used in the dialog.

The screenshot of the Bus Allocation dialog is shown in Figure 27.

A table with the list of currently allocated buses will be shown (Figure 27). The header of the table indicates the meaning of each column, such as bus name and type. Each row in the table represents an allocated bus that is part of the current system target architecture. For each bus, its name, its type, its attributes are shown in the respective columns of the table. The list of allocated buses can be sorted by any column and in ascending or descending order each by clicking into the corresponding column header. By default, the list is sorted by ascending names. Users can perform the following actions:

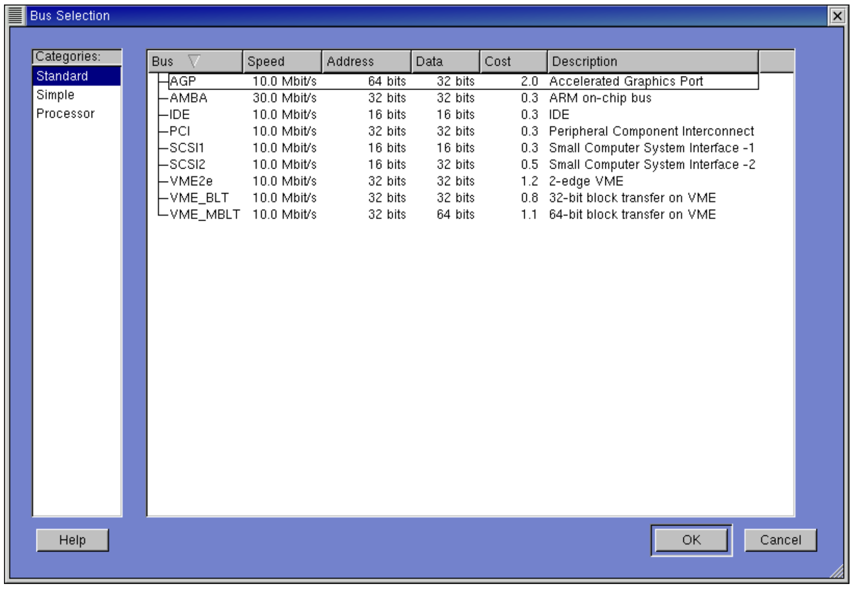
Bus Adding In order to add a bus to the design, users click the button *Add* to pop up the Bus Selection dialog which opens and loads the bus database and allows users to select an additional bus out of the bus database. The screenshot of the Bus Selection dialog is shown in Figure 28.

At the left of the Bus Selection dialog is a bus category table. Each row represents one category of buses in the database. For example, row *Standard* contains all the standard buses in the database.

By clicking and selecting one row in the table at left, users will be shown all the buses in the selected category in the table at the right. Each row of the table at the right represents one type of bus in the database under the selected category. The name of the bus type (*Bus* column) and other attributes of the bus type are displayed in separate columns. Users can select the desired bus type by clicking the corresponding row.

There are two buttons at the bottom of the Bus Selection dialog: *Ok* and *Cancel*. By clicking the *Ok* button, an additional bus of the selected bus type and with an automatically determined name is added into the design's allocated network architecture. In addition, buses can be allocated by double-clicking into the desired bus type row in the Bus Selection dialog (equivalent to selecting the row and pressing *Ok*). Clicking the *Cancel* button aborts and cancels Bus selection without changes to the bus allocation. Either clicking *Ok* or *Cancel* button will close the Bus Selection dialog and return to the Network Allocation dialog.

Bus Copying In order to duplicate an existing bus in the design's bus allocation, users can select a bus by clicking the corresponding row in the allocation table and click the button *Copy*. Clicking *Copy* will add a new bus instance with an automatically determined name and with the same type, attributes and description as the currently selected bus to the design's allocation.



Bus	Speed	Address	Data	Cost	Description
AGP	10.0 Mbit/s	64 bits	32 bits	2.0	Accelerated Graphics Port
AMBA	30.0 Mbit/s	32 bits	32 bits	0.3	ARM on-chip bus
IDE	10.0 Mbit/s	16 bits	16 bits	0.3	IDE
PCI	10.0 Mbit/s	32 bits	32 bits	0.3	Peripheral Component Interconnect
SCSI1	10.0 Mbit/s	16 bits	16 bits	0.3	Small Computer System Interface -1
SCSI2	10.0 Mbit/s	16 bits	32 bits	0.5	Small Computer System Interface -2
VME2e	10.0 Mbit/s	32 bits	32 bits	1.2	2-edge VME
VME_BLT	10.0 Mbit/s	32 bits	32 bits	0.8	32-bit block transfer on VME
VME_MBLT	10.0 Mbit/s	32 bits	64 bits	1.1	64-bit block transfer on VME

Figure 28: Bus Selection dialog.

Bus Deletion . In order to remove a bus from the design's allocation, users can select the target bus to be removed in the allocation table and click the *Remove* button. Clicking *Remove* will remove the selected bus from the list of allocated buses. Buses that currently have PEs or CEs connected to them will not be available for deletion (the *Remove* button will be inactive and grayed out for them).

Bus Editing Users can edit the bus name and bus description in the bus allocation tab by clicking into the respective *Name* or *Description* column of the corresponding bus. Clicking into any of these cells will allow editing of the respective text in the cell by opening a text edit box in place. Pressing the *Esc* key during editing aborts the edit operation. Pressing *Enter* accepts the entered text and changes the bus name or description in the allocation accordingly.

There are two buttons at the bottom of Bus Allocation dialog: *Ok* and *Cancel*. By clicking the *Ok* button, the bus allocation is saved back into the design. If users click the *Cancel* button, Bus Allocation will be cancelled and all modifications to the allocation and connectivity are discarded. Either clicking *Ok* or *Cancel* button will close the Bus Allocation dialog.

Error/Information Messages There are several possible errors during Bus Allocation in general: if no top-level behavior is selected in the design when selecting *Main* ::

Synthesis ⇒ *Allocate Buses* an Error dialog to that effect will be popped up and the Bus Allocation operation will be aborted. Furthermore, clicking the *Ok* button in the Bus Allocation dialog will write the allocation tables back to the design. In case of errors, an Error dialog is popped up and Bus Allocation is aborted.

There are several errors that can happen specifically during bus allocation:

(a) During Bus Editing, if users try to give buses names which is already used as the name of another bus in the design, an Error dialog will be popped up, a corresponding error message will be shown, and the editing operation will be aborted and cancelled.

(b) During Bus Adding, when clicking the *Add* button the bus database stored on disk will be opened and loaded in order to read the list of available buses from the database. In case of errors during database opening (e.g. file errors or wrong file format), an Error dialog will be popped up and the Bus Adding operation will be aborted.

Furthermore, when adding a bus to the allocation via the Bus Selection dialog, the selected bus type is read from the database. In case of database read errors (file errors, database format errors) during this operation, an Error dialog will be popped up and the Bus Adding operation aborted.

Channel Mapping In order to implement the communication in the specification model on the allocated communication architecture consisting of buses, users have to be able to map the channels in the specification onto the allocated Buses. Channel Mapping allows for mapping of channel instances in the design into buses. Channel mapping information is stored as annotations attached to channel definitions inside a behavior class. As a result, multiple incarnations of the same channel in different instances of the parent behavior will all share the same mapping information. Users can also map behaviors to buses. If a channel is not mapped to any bus, all of its instances will be implicitly (and recursively) mapped onto the same bus as the parent behavior class in which they are instantiated in. Note that user must map all the channels under the top-level behavior to buses either implicitly (by mapping the top-level behavior itself) or explicitly.

Communication Refinement Communication Refinement executes the implementation decisions made in the other Communication Synthesis tasks by refining the current Architecture Model into an automatically generated Communication Model based on and reflecting the decision made during Bus Allocation.

Operation In order to do refinement, users can select the *Main* :: *Synthesis* ⇒ *Network Refinement* menu entry to pop up the Communication Refinement dialog. The screenshot of the Communication Refinement dialog is shown in Figure 29.

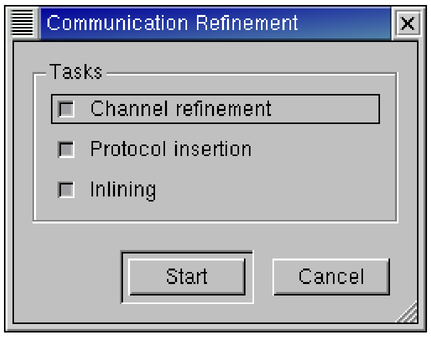


Figure 29: Communication Refinement dialog.

In the Communication Refinement dialog, users can select whether individual sub-tasks of the communication refinement process will be performed or not. By checking or unchecking the check boxes tasks are turned on and off and partially or completely refined models can be generated. By default, all tasks are turned on. The three sub-tasks of network refinement are:

(a) *Channel refinement* which refines the architecture model to the communication model.

(b) *Protocol insertion* which generates implementations contains protocol layers.

(c) *Inlining* which inline the protocol layers to PEs.

If all tasks are turned off, no output model is generated and only input validation is performed. In other cases, exactly one output model at varying levels of refinement is generated.

User can then start the communication refinement process by clicking the *Start* button. If users click the *Cancel* button, the Communication Refinement operation will be aborted and cancelled. Either clicking *Start* or *Cancel* buttons will close the Communication Refinement dialog.

After clicking the *Start* button, the communication refinement command line components will be executed in the background. Any diagnostic, status and informative output of the network refinement tools will be shown in the Refinement tab of the Output Window ($Output :: Refine$).

When the communication refinement process is finished, the newly generated network model is automatically opened and loaded, and a corresponding new Design Window is created in the Workspace. The new Design Window is automatically activated and raised to the front. In addition, the new communication model is automatically added to the current project (see Design Adding, Section 4.2.5) as a child of the architecture model it was generated from.

While the communication refinement background tools are running, the majority of the main SCE GUI is disabled. However, users can abort/kill execution of the background tools by selecting *Main* :: *Synthesis* ⇒ *Stop*. After clicking, the current communication refinement background task is aborted.

Error/Information Messages If the communication refinement background tools abort with an error or are killed via the *Main* :: *Synthesis* ⇒ *Stop* menu entry, an Error dialog with a corresponding error message will pop up. Specifically, the communication refinement background tools check for and can produce the following classes of errors:

- No top level behavior.
- No or invalid bus allocation information.

Upon confirming the error, the remainder of the Communication Refinement Operation will be cancelled.

If the design the new model was generated from is not in the project, the new model is not added to the project and an Error dialog to that effect will be popped up.

4.5.3 Decision Import

In order to take over implementation decisions from a previously done design into a new design, SCE allows to import design decisions from one design into another. With this functionality, previously made design decisions can easily be transfered to a new design as a starting point for synthesis.

Operation Users can import design decisions from any currently opened and loaded design into the currently active design by selecting the Main Menu command *Main* :: *Synthesis* ⇒ *Import Decisons*. As a result, the Import Decisions dialog will be popped up as shown in Figure 30. The solid rectangular with triangle symbol represents the drop-down menu. The square with check mark represents a select box. The round rectangulars represents buttons.

First, in order to select a design from which decisions are imported, users select the name of the source design in the *Source design* drop-down box. The combo box contains all the names of currently opened Design Windows except for the currently active one. Users select one design from them as the source design.

Secondly, users can select decisions which will be imported to the current design. Users do it by checking the selection items as follows:

PE Allocation Copies the allocated PEs/memories from the source design to the currently active design. For all behaviors in the source design, PE allocation annotations are copied to the behavior (if any) with the same name in the target design.

Behavior Mapping Copies behavior mapping information from the source design to the currently active design. For all behaviors in the source design, PE mapping annotations are copied to the behavior (if any) with the same name in the target design.

Variable Mapping Copies variable mapping information from the source design to the currently active design. For all variable definitions in all behaviors in the source design, PE mapping annotations are copied to the variable (if any) with the same name in the behavior (if any) with the same name in the target design.

Bus Allocation Copies the allocated busses from the source design to the currently active design. For all behaviors in the source design, bus allocation annotations are copied to the behavior (if any) with the same name in the target design.

Channel Mapping Copies the channels to buses mapping decision

The Import Decisions dialog includes an additional check item to select whether any existing annotations in the target design should be overwritten or kept.

There are two buttons on the Import Decisions dialog: *Import* and *Cancel*. By clicking the *Import* button, the design decisions of the selected items are imported to the selected design from the source design. If users clicks the *Cancel* button, the action of Decision Importing will be aborted and cancelled. Either clicking *Import* or *Cancel* button will close the Import Decisions dialog.

Error/Information Messages Decision Importing requires reading and writing of annotations in the designs. In case of errors (e.g. wrong annotation format) upon pressing the *Import* button, an Error dialog will be popped up and the Decision Importing operation will be aborted.

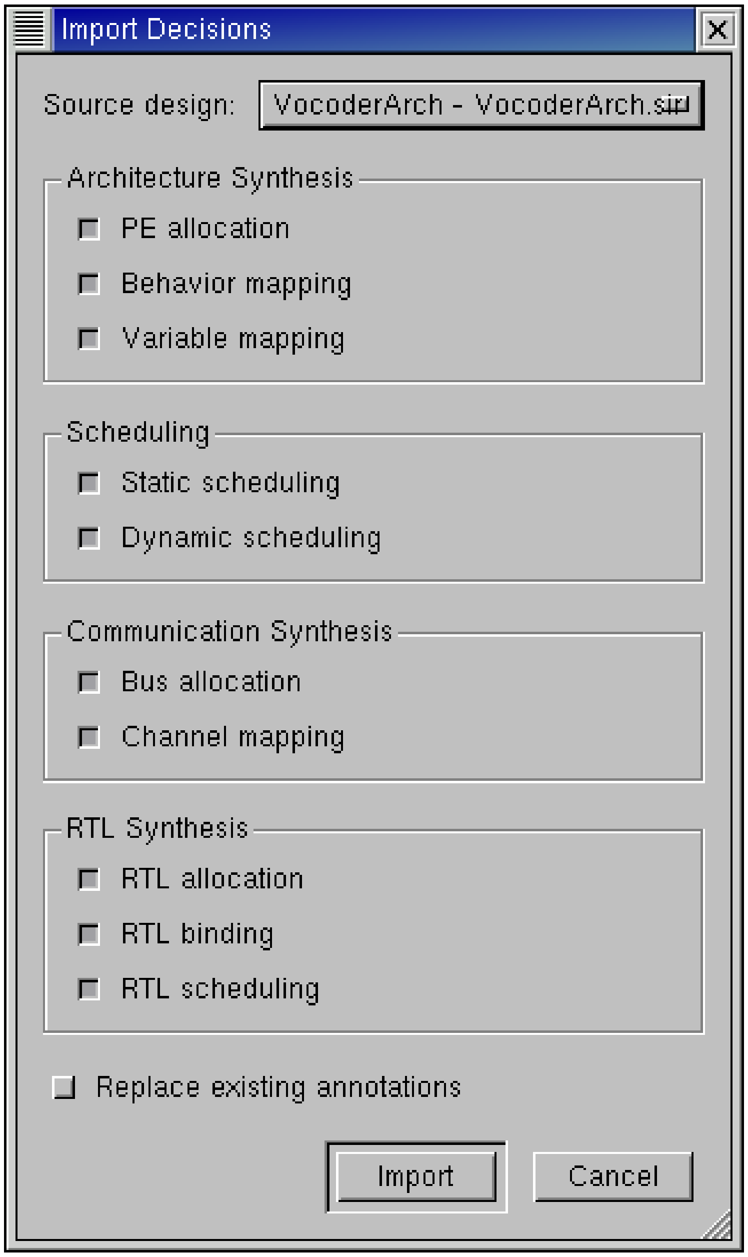
4.6 Window Management

Window Management deals with the management of design windows in the Workspace. Window Management allows for closing, resizing, and arranging of multiple simultaneously opened design windows within the Workspace. Specifically, the tasks for Window Management are:

Window Closing Users can close the currently active design window in the Workspace by selecting $Main :: Window \Rightarrow Close$. In addition, any of the design windows can be closed by clicking on a respective icon in the window's title bar.

Users can close all the currently opened design windows in the Workspace by selecting $Main :: Window \Rightarrow Close\ All$.

In all cases, closing a design window triggers a File Closing action for the corresponding design file (see Section 4.3.3).



Import Decisions

Source design: VocoderArch - VocoderArch.sir

Architecture Synthesis
- [✓] PE allocation
- [✓] Behavior mapping
- [✓] Variable mapping

Scheduling
- [✓] Static scheduling
- [✓] Dynamic scheduling

Communication Synthesis
- [✓] Bus allocation
- [✓] Channel mapping

RTL Synthesis
- [✓] RTL allocation
- [✓] RTL binding
- [✓] RTL scheduling

- [ ] Replace existing annotations

Figure 30: Import Decisions dialog.

Window Arranging Users can automatically arrange design windows in the Workspace in a variety of manners. Selecting $Main :: Window \Rightarrow Tile$ will rearrange the design windows in the Workspace in a tiled fashion. Selecting $Main :: Window \Rightarrow Cascade$ will rearrange the design windows in the Workspace in a cascaded manner. Apart from that, windows can be freely resized and moved within the Workspace by dragging their title bar or borders. In addition, users can maximize and minimize design windows by clicking on a respective icon on the window's title bar.

Window Switching Selecting $Main :: Window \Rightarrow Next$ or $Main :: Window \Rightarrow Previous$ will switch the focus to and activate the next/previous design window in the list of opened windows. Using these actions, users can cycle through the list of windows. Design windows are ordered in the window list according to the order in which they were opened. In addition, users can activate and raise any of the opened design windows by clicking into the window.

Finally, the bottom of the $Main :: Window$ menu contains entries for all currently opened design windows. Selecting any of these menu entries will activate and raise the corresponding design window.

Window Toggling Selecting $Main :: Window \Rightarrow Project\ Manager$ or $Main :: Window \Rightarrow Output\ Window$ will toggle (turn on and off) displaying of Project and Output Windows, respectively.

References

[1] R. Dömer, A. Gerstlauer and D. D. Gajski. *SpecC Language Reference Manual, Version 2.0,* SpecC Technology Open Consortium (STOC), Japan, December 2002.

[2] R. Dömer. *The SpecC Internal Representation (V2.0.3),* Technical Report 03-21, University of California, Irvine, January 1999.

[3] J. Peng, A. Gerstlauer, K. Ramineni, R. Dömer and D. D. Gajski. *System-On-Chip Specification Style Guide,* Technical Report CECS-TR-03-21, Center for Embedded Computer Systems, University of California, Irvine, 2003.

A Manual Pages

This appendix contains the documentation in the form of manual pages for external, third-party tools used by SCE.

A.1 `scc` - SpecC Compiler

NAME

scc – SpecC Compiler

SYNOPSIS

scc *–h*

scc *design* [*command*] [*options*]

DESCRIPTION

scc is the compiler for the SpecC language. The main purpose of **scc** is to compile a SpecC source program into an executable program for simulation. Furthermore, **scc** serves as a general tool to translate SpecC code from various input to various output formats which include SpecC source text, SpecC binary files in SpecC Internal Representation format, and other compiler intermediate files.

Using the first command syntax as shown in the synopsis above, a brief usage information and the compiler version are printed to standard output and the program exits. Using the second command syntax, the specified *design* is compiled. By default, **scc** reads a SpecC source file, performs preprocessing and builds the SpecC Internal Representation (SIR). Then, C++ code is generated, compiled and linked into an executable file to be used for simulation. However, the subtasks performed by **scc** are controlled by the given *command* so that, for example, only partial compilation is performed with the specified *design*.

On successful completion, the exit value 0 is returned. In case of errors during processing, an error code with a brief diagnostic message is written to standard error and the program execution is aborted with the exit value 10.

For preprocessing and C++ compilation, **scc** relies on the availability of an external C++ compiler which is used automatically in the background. By default, the GNU compiler **gcc/g++** is used.

ARGUMENTS

design	specifies the name of the design; by default, this name is used as base name for the input file and all output files;

COMMAND

The *command* has the format **-** *suffix1* 2 *suffix2*, where *suffix1* and *suffix2* specify the format of the main input and output file, respectively. This command also implies the compilation steps being performed. By default, the command –sc2out is used which specifies reading a SpecC source file (e.g. design.sc) and generating an executable file (e.g. a.out) for simulation. All necessary intermediate files (e.g. design.cc, design.o) are generated automatically.

Legal command suffixes are:

sc	SpecC source file (default: *design.sc)*
si	preprocessed SpecC source file (default: *design.si)*
sir	binary SIR file in SpecC Internal Representation format (default: *design.sir)*
cc	C++ simulation source file (default: *design.cc)*
h	C++ simulation header file (default: *design.h)*
cch	both, C++ simulation source file and C++ header file (default: *design.cc* and *design.h)*
o	linker object file (default: *design.o)*
out	executable file for simulation (default: *design);* however, with the –ip option, a shared library will be produced (default: *libdesign.so)*

OPTIONS

–v | *–vv* | *-vvv*	increase the verbosity level so that all tasks performed are logged to standard error (default: be silent); at level 1, informative messages for each task performed are displayed; at level 2, additionally input and output file names are listed; at level 3, very detailed information about each executed task is printed;
–w | *–ww* | *-www*	increase the warning level so that warning messages are enabled (default: warnings are disabled); four levels are supported ranging from only important warnings (level 1) to pedantic warnings (level 4); for most cases, warning level 2 is recommended (–ww);

–g enable debugging of the generated simulation code (default: no debugging code); this option disables optimization;

–O enable optimization of the generated simulation code (default: no optimization); this option disables debugging;

–ip enable intellectual property (IP) mode; when generating a SIR binary or SpecC text file, only declarations of symbols marked public will be included (the public interface of an IP is created); when generating C++ code, non-public symbols will be output so that they will be invisible outside the file scope; when compiling or linking, the compiler and linker are instructed to create a shared library instead of an executable file (creation of an IP simulation library);

–n suppress creation of new log information when generating the output SIR file (default: update log information); see also section ANNOTATIONS below;

–sl suppress source line information (preprocessor directives) when generating SpecC or C++ source code (default: include source line directives);

–sn suppress all annotations when generating SpecC source code (default: include annotations);

–st tabulator stepping set the tabulator stepping for SpecC/C++ code generation; this setting is used for code indentation; a value of 0 will disable the indentation of the generated code (default: 4);

–sT system tabulator stepping set the system tabulator stepping (\t) for SpecC/C++ code generation; if set, tab characters will be used for indentation; if a value of 0 is specified, only spaces will be used for indentation (default: 8);

–sw line wrapping set the column for line wrapping; in code generation, any line longer than this value is subject to line wrapping; if a value of 0 is specified, no line wrapping will be performed (default: 70);

–i input file specify the name of the input file explicitly (default: *design.suffix1);* the name '-' can be used to specify reading from standard input;

–o output file	specify the name of the final output file explicitly (default: *design.suffix2);* the name ’-’ can be used to specify writing to standard output;
–D	do not define any standard macros; by default, the macro __SPECC__ is defined automatically (it is set to 1); furthermore, implementation dependent macros may be defined; this option suppresses the definition of all these macros;
–Dmacrodef	define the preprocessor macro *macrodef* to be passed to the preprocessor;
–U	do not undefine any macros; by default, few macros are undefined automatically (in order to allow C/C++ standard header files to be used); this option is implementation dependent;
–Uundef	undefine the preprocessor macro *undef* which will be passed to the preprocessor as being undefined; the macro *undef* will be undefined after the definition of all command-line macros; this allows to selectively suppress macros from being defined in the preprocessing stage;
–I	clear the standard include path; by default, the standard include path consists of the directory $SPECC/inc; this option suppresses the default include path;
–Idir	append *dir* to the include path (extend the list of directories to be searched for including source files); include directories are searched in the order of their specification; unless suppressed by option –I, the standard include path is automatically appended to this list; by default, only the standard include directories are searched;
–L	clear the standard library path; by default, the standard library path consists of the directory $SPECC/lib; this option suppresses the default library path;
–Ldir	append *dir* to the library path (extend the list of directories to be searched for linker libraries); the library path is searched in the specified order; unless suppressed by option –L, the standard library path is automatically appended to this list; by default, only the standard library path is searched;

–l when linking, do not use any standard libraries; the default libraries are displayed when calling the compiler with the –h option; the –l option suppresses linking against theses standard libraries;

–llib pass *lib* as a library to the linker so that the executable is linked against *lib;* libraries are linked in the specified order; unless suppressed by option –l, the standard libraries are automatically appended to this list; by default, only standard libraries are used;

–P reset the import path; clear the list of directories to be searched for importing files; by default, the current directory is searched first, followed by the standard import directory $SPECC/import; this option suppresses this standard import path;

–Pdir append *dir* to the import path, extending the list of directories to be searched for importing files; import directories are searched in the order of their specification; unless suppressed by option –P, the standard search path is automatically appended to this list; by default, only the standard import path is searched;

–xpp preprocessor_call redefine the command to be used for calling the C preprocessor (default: ”g++ -E -x c %p %i -o %o”); the preprocessor call must contain three markers %p, %i and %o, which indicate the options and file names used in the call; in the specified string, the %p marker will be replaced with the list of specified preprocessor options; the %i and %o markers will be replaced with the actual input and output filenames, respectively;

–xcc compiler_call redefine the command to be used for calling the C/C++ compiler (default: ”g++ -c %c %i -o %o”); the compiler call must contain three markers %c, %i and %o, which indicate the options and file names used in the call; in the specified string, the %c marker will be replaced with the list of specified compiler options; the %i and %o markers will be replaced with the actual input and output filenames, respectively;

–xld linker_call redefine the command to be used for calling the linker (default: ”g++ %i -o %o %l”); the linker call must contain three markers %l, %i and %o, which indicate the options and file names used in the call; in the specified string, the %l marker will be replaced with the list of specified linker options; the %i and %o markers

will be replaced with the actual input and output filenames, respectively;

–xp preprocessor_option pass an option directly to the C/C++ preprocessor (default: none);

–xc compiler_option pass an option directly to the C/C++ compiler (default: none);

–xl linker_option pass an option directly to the linker (default: none);

ENVIRONMENT

SPECC is used to determine the installation directory of the SpecC environment where SpecC standard include files (directory $SPECC/inc), SpecC standard import files (directory $SPECC/import), and SpecC system libraries (directory $SPECC/lib) are located.

SPECC_LICENSE_FILE determines the license file (path and file name) to be used by the SpecC environment; if undefined, the environment variable *SPECC* is used as the path to the license file called ”license.sce”; if neither *SPECC_LICENSE_FILE* nor *SPECC* exist, the file ”license.sce” is searched in the current directory;

ANNOTATIONS

The following SpecC annotations are recognized by the compiler:

_SCE_LOG contains the log information of the SIR file; this global annotation is created and maintained automatically by the SpecC compiler and the SpecC tool set and can be used to determine the origin and the operations performed on the design model; *_SCE_LOG* is a composite annotation consisting of a list of log entries, ordered by time of creation; each log entry consists of a time stamp, command line, source file, version info, and an optional comment;

_SCC_RESERVED_SIZE for external behaviors and channels (IP components), this indicates the size reserved in the C++ class for internal use; the annotation type is unsigned int; if found at class definitions, this annotation is checked automatically for reasonable values; for IP declarations, the annotation can be created automatically with the –ip option;

_SCC_PUBLIC for global symbols, this annotation indicates whether the symbol is public and will be visible in a shared library; the annotation type is bool; this annotation only is recognized with the –ip option;

VERSION

The SpecC compiler **scc** is version 2.2.0.

AUTHOR

Rainer Doemer <doemer@ics.uci.edu>

COPYRIGHT

(c) 1997-2004 CECS, University of California, Irvine

SEE ALSO

gcc(1), **g++**(1), **sir_delete**(l), **sir_depend**(l), **sir_import**(l), **sir_isolate**(l), **sir_list**(l), **sir_note**(l), **sir_rename**(l), **sir_strip**(l), **sir_tree**(l), **sir_wrap**(l)

BUGS, LIMITATIONS

Variables of enumerator type cannot be initialized at the time of their declaration. The SpecC compiler issues a (false) error message in this case. As a simple work-around, however, enumerator variables can be initialized by use of standard assignment statements at the beginning of their lifetimes.