The SpecCharts/SpecSyn User's Manual
Version 3.0b

UC Irvine - CADLAB

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717, USA
Tel : (714) 856-8059
Fax : (714) 856-4056

February 9, 1994

Frank Vahid (vahid@ics.uci.edu)
Sanjiv Narayan (narayan@ics.uci.edu)

Contents

1 Software installation and setup **3**

1.1 SpecCharts/SpecSyn software synopsis 3

1.2 Installation 3

1.3 Setting up 5

2 SpecCharts Tutorial **6**

2.1 A SpecCharts example 6

2.2 Editing/viewing SpecCharts with xspeccharts 7

2.3 Translating SpecCharts to VHDL 9

2.4 Entering a new SpecCharts specification 9

2.5 Checking syntax 10

3 Estimators tutorial **11**

3.1 Estimating design clock cycle 11

3.2 Estimating design area 11

4 SpecSyn Tutorial **12**

4.1 System design with SpecSyn 12

5 Bypassing X windows **14**

6 SpecCharts tools **15**

6.1 xspeccharts 15

6.1.1 SYNOPSIS 15

6.1.2 DESCRIPTION 15

6.1.3 MENU OPTIONS 16

7 xspecsyn **17**

7.1 SYNOPSIS 17

7.2 DESCRIPTION 17

7.3 NOTES 23

7.4 OPTIONS 23

8 References **24**

A FTP instructions for SpecSyn/SpecCharts **25**

B SpecCharts application notes **26**

C SpecSyn application notes **28**

1 Software installation and setup

1.1 SpecCharts/SpecSyn software synopsis

Platforms tested on:	SUN-4 with SunOS 4.1.2
Graphics:	X Windows X11R5
Disk requirements:	20 Mbytes
Expiration date:	See attached software release form

1.2 Installation

To load the software, complete the following steps:

1. Insert the SpecSyn Distribution Tape into the tape drive. If you do not have a tape, you may aquire the software via ftp. See Appendix A.

2. Select a directory for installing the SpecSyn software. We shall refer to the pathname of this directory by *instdir*. Make sure that *instdir* has the appropriate write permissions. Move to the installation directory.

   ```
   %   cd instdir
   ```

3. The SpecSyn software will be loaded into a subdirectory named `specsyn`. If a file or directory named `specsyn` already exists in *instdir*, it will be overwritten. To save an existing file or directory named `specsyn`, move it to a different directory or rename it.

4. Load the SpecSyn software by typing,

   ```
   %   tar xvf /dev/rst8
   ```

5. You can check to see if the installation was successful by listing the files that were loaded. The following files should appear in response to the `ls` command,

   ```
   %   ls -l specsyn
   .......   README
   .......   XSpecsyn
   .......   bin/
   .......   demo/
   .......   doc/
   .......   man/
   ```

6. Remove the tape from the tape drive.

We now briefly describe the files/directories in the `specsyn` directory that were loaded from the tape. SpecCharts is the language in which system specifications are captured. The SpecSyn system-level synthesis tool operates on these SpecCharts to perform a number of synthesis tasks.

1. **README** : This file briefly explains the file setup for this software release.

2. **XSpecsyn** : This file contains X application defaults for the `xspecsyn`, `xspeccharts`, `xscestarea`, and `xscestclk` X programs. You should merge `XSpecsyn` with the X resource database before running any of the SpecSyn X programs by typing :

```
%   xrdb -merge XSpecsyn
```

3. **bin** : This directory contains the SpecSyn executables. Further details can be found in the *man* pages.

`xspeccharts` :	X interface to the SpecCharts edit, view, and transform tools.
`sc2vhdl` :	SpecCharts to VHDL translator.
`sccheck` :	A SpecCharts syntax checker.
`scflat` :	Flattens hierarchy in SpecCharts.
`scs2c` :	Sequential to concurrent SpecCharts behavior conversion.
`sctree` :	Generates a tree representing the SpecCharts hierarchy.
`xspecsyn` :	X-based system-design tool.
`scestarea` :	Area estimator.
`xscestarea` :	X interface to `scestarea`.
`scestclk` :	Clock cycle estimator.
`xscestclk` :	X interface to `scestclk`.
`scestperf` :	Performance estimator.

Note: all executables require a SpecCharts file to end with `.sc`, and all VHDL packages used by a SpecCharts file to end with `.vhd`.

4. **demo** : This directory contains a tutorial for learning about SpecCharts and SpecSyn.

5. **doc** : This directory contains the following files :

   - `user_manual.ps` : Postscript version of this user's manual.

6. **man** : This directory consists of the manual pages for each of the above executables. These man pages are also included in this user's manual as Sections 6 and 7. You can install these man pages in the online manual of your system by updating your MANPATH environmental variable:

```
%   setenv MANPATH=( instdir/specsyn/man:$MANPATH )
```

   Then type `rehash` (to recompute the internal UNIX hash table of the contents of your directories to account for the updated MANPATH). The man pages can then be viewed using the `man` command. For example, after updating the MANPATH as above, the man page for `sc2vhdl` can be viewed by typing:

```
%   man sc2vhdl
```

If the man page is not found, try sourcing your .cshrc. The man pages can also be viewed directly by using the `nroff` command. For example, to view the man page for `sc2vhdl` from the `specsyn` directory, type :

```
%   nroff  -man  man/manl/sc2vhdl.l | more
```

1.3 Setting up

The simplest approach to using the SpecCharts and SpecSyn tools is through the X interface tools `xspeccharts` and `xspecsyn`, because you can activate every other tool by selecting a menu item. Alternatively, you can activate all tools from the UNIX command line, as described in the *man* pages. To setup your system to run `xspeccharts` and `xspecsyn`, first start X windows. Then add the *instdir*`/specsyn/bin` to your path, merge the SpecSyn graphics application default file with your X resource database, and set the X key symbols environment variable.

```
%   set path=(instdir/specsyn/bin    $path)
%   xrdb -merge instdir/specsyn/XSpecsyn
%   setenv XKEYSYMDB instdir/specsyn/XKeysymDB
```

SHORTCUT: After setting the SPECSYN_DIR variable in the file *instdir*`/specsyn/setup` to *instdir*`/specsyn`, just type: `source` *instdir*`/specsyn/setup`.

2 SpecCharts Tutorial

This section presents a simple example captured with **SpecCharts** and the application of system-design tasks using the **SpecSyn** system-design tool. The example demonstrates use of `xspeccharts` and `sc2vhdl`. Details of the SpecCharts language, its translation to VHDL, and various examples can be found in [1, 2, 3, 4].

2.1 A SpecCharts example

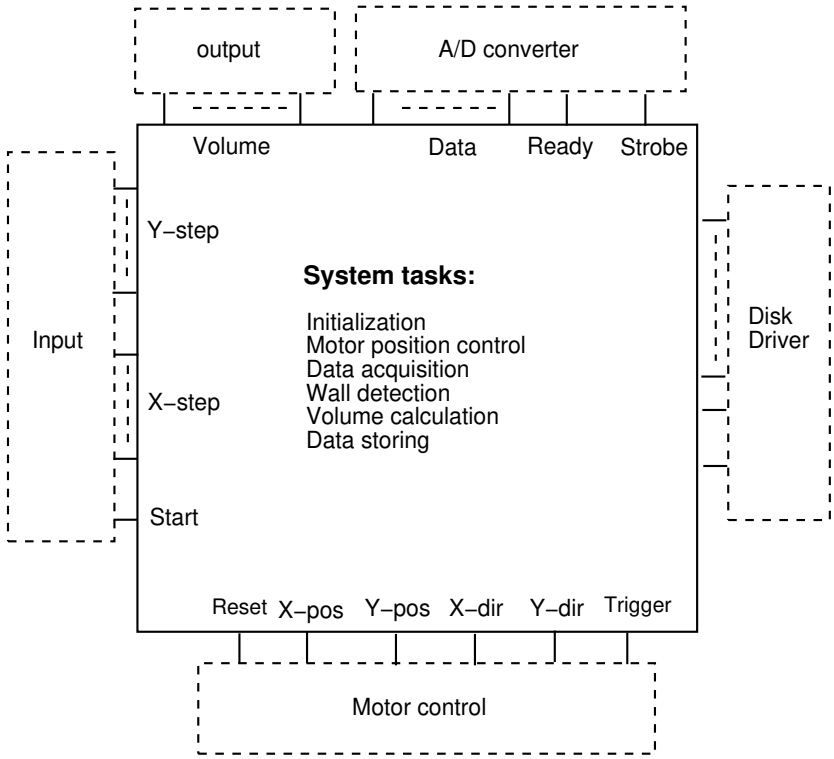

Figure 1: The volume measurer system's environment

Figure 1 shows the environment of a medical system which measures the volume of a bladder by positioning and monitoring several sensors. The textual SpecCharts specification of this system is stored in the file `vol.sc` in the `demo` directory. The specification consists of a hierarchy of behaviors. Each behavior's functionality is described either with VHDL sequential statements (a **leaf** behavior), with **concurrent** subbehaviors, or with **sequential** subbehaviors sequenced by arcs. A **TOC** are (transition-on-completion) is transitioned when the source behavior completes and the arc condition is true, whereas a **TI** arc (transition-immediately) is transitioned immediately when the arc condition is becomes true. Each behavior may contain VHDL declarations whose scope is all descendant subbehaviors.

x_step and *y_step* are two of the system's inputs, while *x_pos* and *y_pos* are two of its outputs. The entire system is described as a behavior *VolumeSystem*, which is decomposed into sequential subbehaviors *Initiate, Empty, MotorControl1, Output, Empty2, IncY,* and *Xbody.* Each of those subehaviors is a leaf, except for *Xbody* which itself is decomposed into sequential subbehaviors *MotorControl2, DataAcquisition, FlatLevel, AnteriorWall, PosteriorWall, ComputeVolume, Disk,* and *IncX.* Note that *VolumeSystem* contains declarations of a type and of several variables global to all behaviors. Other behaviors declare local variables, such as *temp_index* declared in *DataAcquisition.*

We now describe the functionality of the volume system, although an understanding of the functionality is not required for the tutorial. The volume system controls a transducer, which is attached to a motor, to scan the related abdominal area along two-dimensional grids. At each scanning point, the transducer sends a ultrasonic wave directly into the anatomical region to be examined. When the ultrasonic wave strikes tissues of different acoustic impedance, an echo is reflected back to the transducer. Two major peaks will be generated by the echoes from the anterior and posterior walls of the bladder. Therefore the distance between the anterior and the posterior walls of each section of the bladder can be determined and thus the volume of the bladder can be computed.

Basically, the system consists of six tasks: (1) initialization; (2) motor position control; (3) data acquisition; (4) wall detection; (5) volume calculation; (6) data storing. In the initialization stage, the system loads the number of control steps along x direction and y direction and resets the motors to the initial scanning position. When the starting signal is set, the scanning process begins. The motor position controller sends out the control signals that drive the motors to move the transducer to the scanning position. Then the motor controller activates the transducer to send the ultrasonic wave to the examing point. The data acquisition module converts the ultrasonic echo into digital signals and store them temporarily. The wall detection and volume calculation modules determine the distance between the anterior and posterior walls of the bladder and the volume based on the fetched data. Finally the data storing module stores the fetched data into disk for later analysis. The process continues until the system covers the entire given scanning range.

There are two main imposed performance constraints for this system: (1) data acquisition and conversion of 1000 data points is 1 ms; (2) maximum time between two scans, including time for motor control, data acquisition, wall detection, volume calculation and data storing, is two seconds.

2.2 Editing/viewing SpecCharts with xspeccharts

`xspeccharts` simplifies SpecCharts capture by allowing you to specify each behavior by filling in a table's fields, thus reducing the amount of syntax you must remember and the amount of typing. It also pops up the tables in a hierarchical manner, making the hierarchy of SpecCharts easier to explore than when using a standard text editor such as vi or emacs.

Move to the demonstration directory and activate the tool:

```
%   cd instdir/specsyn/demo
%   xspeccharts &
```

The `xspeccharts` window should appear. At the top of the window are pull-down menus which allow

you to:

- Open, save and syntactically check SpecCharts files (see `xspeccharts` man page).
- Translate SpecCharts to VHDL for simulation (see `sc2vhdl` man page).
- Flatten the SpecCharts hierarchy of behaviors into sequential statements (see `scflat` man page).
- Inline simple procedures (see `scinline` man page).
- Replace enumerated types by constants (see `scenum` man page).
- Convert a sequential behavior to a top-level concurrent one (see `scs2c` man page).

The remainder of this tutorial is identical to the `XSPECCHARTS_DEMO` file found in the demo directory.

1. **Select File => Open** to open the volume measuring system. Select `vol.sc` by clicking on it with your mouse once and then selecting "OK", or by clicking on it twice quickly.

   The left side of the `xspeccharts` window now shows the current file, `vol.sc`, and alphabetically lists the file's behaviors. The declarations and program sections of the currently selected behavior are displayed on the right side of the window, for viewing only and not for editing.

2. **Select** *VolumeSystem* in the behavior list to make that behavior the currently selected one, in order to view that behavior's contents on the right side of the window.

3. **Select "Edit"** to edit the *VolumeSystem* behavior in a new window (or double-click on *VolumeSystem*). The new window divides the behavior into four parts: *Name, Declarations, Program type*, and *Program contents*. To edit any of these, place the cursor at the desired position and click, and then type the desired text. Basic emacs editing commands are available.

4. **Select** *Initiate* **in the** *VolumeSystem* **window** by clicking on it twice quickly. *Initiate* gets highlighted.

5. **Select "Edit Subbehavior"** to open a new window for *Initiate*.

6. **Add** the VHDL declaration "`signal temp : integer ;` " to the declarations part of the *Initiate* window.

7. **Select "File => Save"** in the xspeccharts window to save the new SpecCharts specification in `vol.sc`.

8. **Select "Close"** in the *VolumeSystem* window. This will close the window and all its children windows.

9. **Select File => Quit** (or control-Q) to quit the program.

2.3 Translating SpecCharts to VHDL

1. Start `xspeccharts` and open file `vol.sc` as above. (SHORTCUT – type: `xspeccharts vol.sc &`).

2. **Select Translators => VHDL** from the menu bar. This will translate the SpecCharts file to a simulatable VHDL file. The **sc2vhdl** translator [3, 5] creates the following files:

   - `vol.vhd` which contains a single entity. The entity will have the SpecCharts root-state name with E appended, i.e., VolumeSystemE. The entity will have port declarations identical to the port declarations in `vol.sc`.
   - `vol_sim.vhd` which is a skeleton VHDL file for simulation, generated only if it does not already exist. This file instantiates the entity *VolumeSystemE* so that you can provide stimuli to its ports. It defines a new entity *E*, which can then be simulated. In case the SpecCharts example converted to VHDL does not have any ports, the file `vol_sim.vhd` can be ignored.

   An additional file `vol_sim_demo.vhd` has been provided with the release and provides a sample set of stimuli for the volume measurer system.

3. In another window, but in the same directory, compile your VHDL files in the proper order as shown below, and then simulate entity *E*. Here we shall be using `vol_sim_demo.vhd` to provide stimuli. You could instead use `vol_sim.vhd` by adding your own stimuli to that file.

   ```
   %    analyze   vol.vhd
   %    analyze   vol_sim_demo.vhd
   %    simulate   -i cmds.com   -t ns   E
   ```

   where *analyze* and *simulate* are specific to your VHDL environment. `cmds.com` contains simulation commands specific to the Synopsys simulator. You need to modify it for your simulator. The `-t ns` option indicates to the Synopsys simulator that the smallest time unit is nanoseconds.

2.4 Entering a new SpecCharts specification

In this section, we will enter a SpecCharts specification which adds the integers from 1 to a specified limit (say, 10).

1. Start `xspeccharts`.

2. **Select File => New** from the menu bar. This will prompt you for a new file name. Enter `new.sc` and click "Ok".

3. Type the name *Top* in the "Current Behavior" window to add a new behavior with this name.

4. **Select "Edit"** to add create a window for *Top*. Add the following to the declarations section:

   ```
   variable limit :  integer ;
   variable sum :  integer := 0 ;
   ```

Set the behavior *type* to "Sequential Substates". Add the following to the program section for *Top*:

```
InitState :  (TOC, true, AddState) ;
AddState :  (TOC, true, stop) ;
```

Thus we are specifying that *Top* consists of two sequential subbehaviors *InitState* and *AddState*.

5. **Select** *InitState*, **click "Edit Substate"** to open a new window.

6. In the *InitState* window, set the program type to "Code" to specify that *InitState* is a leaf behavior with VHDL code. Add the following to the program section:

```
limit := 10 ;
```

7. **Select "Save" followed by "Close"** in the *InitState* window. This action saves the entire specification in a file. You can also save by selecting **File** => **Save** from the top-level `xspeccharts` window.

8. In a similar manner, open a window for leaf behavior *AddState* and add the following to the program section:

```
for i in 1 to limit loop
    in sum := sum + i ;
end loop ;
```

Save, and then close the *AddState* and *Top* windows.

9. The SpecCharts specification with root behavior *Top* will be stored in file `new.sc` which can be simulated as explained earlier.

2.5 Checking syntax

After you have opened a SpecCharts file or created a new specification, you can verify syntax by selecting **File** => **Check** from the top-level menu bar. A message will indicate if the syntax check was successful or will point to the line in the SpecCharts file where an error exists.

3 Estimators tutorial

This section presents examples of SpecCharts clock and area estimators, `xscestclk` and `xscestarea`.

3.1 Estimating design clock cycle

This section demonstrates use of the `xscestclk` tool which, given a specification and an RT-component file, helps determine a clock cycle to use in an RT design [6].

1. Type `xscestclk vol.sc`. The `xscestclk` window appears with the SpecCharts file field set to `vol.sc`. We will use the default component file, `vti100.cmp` , to get the RT-component delays.
2. **Select "Maximum operator delay"** to estimate the the clock cycle as the largest operator delay. The frequency of occurrence of operators and their average utilization is displayed.
3. **Select "Slack minimization"** to estimate the the clock cycle using the Slack Minimization method. The frequency of occurrence of operators and their average utilization is displayed.
4. **Select "Quit"** to exit.

3.2 Estimating design area

This section demonstrates use of the `xscestarea` tool which, given a specification and an RT-component file, estimates the area of an RT-level design that might be synthesized from the specification. [7].

1. Type `xscestarea vol.sc`. The `xscestarea` window appears with the SpecCharts file field set to `vol.sc`. We will again use the default component file.
2. **Enter** *VolumeSystem* as the "State Name" for which we wish to estimate area.
3. **Click "Estimate"** to start the estimator. The estimator will compute and display the components in the design; the microstates in each SpecCharts behavior; the number of registers, memories, muxes and operators; and finally, the area of the control logic, memories and the datapath.

   You may enter another behavior and estimate its area by entering the behavior name and then clicking "Estimate".
4. **Select "Quit"** to exit.

4 SpecSyn Tutorial

This section presents the application of system-design tasks using the **SpecSyn** system-design tool on a given example. Details of the system-design methodology which SpecSyn supports can be found in [8].

4.1 System design with SpecSyn

This section describes use of the `xspecsyn` tool which, given a specification, aids a designer in allocating system components such as processors and ASICs, in partitioning the specification among those components such that constraints are satisfied, and in refining the specification into a new one containing the allocation and partition information.

Move to the demonstration directory and activate the tool:

```
%   cd instdir/specsyn/demo
%   xspecsyn vol.sc &
```

The `xspecsyn` window should appear. The remainder of this tutorial is identical to that found the *instdir*`/specsyn/demo/XSPECSYN_DEMO`.

1. **Choose the functional objects** to be partitioned, which determines the partitioning granularity.
   (a) Select "Display => Objects => Choose"
   (b) In the new window, select "ReadFromFile", then in the file dialog select "OK" for file vol.objs. Scroll through the list of objects to see which objects are chosen. An unselected object is treated as part of its parent object.
   (c) Select "OK" Estimation information is built and the main SpecSyn display is filled in. The main display contains a summary of the current design status. Currently the design is just a set of functional objects.
2. **Open the Partitioner/Allocation window**.
   (a) Select "Partition/Allocate => Open Window" By default, each channel is initially mapped to its own bus, where the width and bitrate of each bus are unconstrained. This mapping is done automatically because it is normally the first step designers do. It can be undone by selecting "Allocate => Delete-class", and then selecting "bus" in the dialog.
3. **Allocate** two chips.
   (a) Select "Library => Item => Select"
   (b) Select "asictype2" in the dialog, then select "OK". Note that this chip type has an area and pin constraints of 10,000 and 100, and costs $60.

(c) "Select Allocate => Add", then select "OK" in the dialog to confirm the chip's name.

(d) Select "Allocate => Add", again, and select "OK". Note that two chips, chip1 and chip0, have been added to the main display. Since no objects have yet been mapped to these chips, the area and pin estimates are 0. The area and pin constraints are shown just after these estimates.

4. **Partition** the functional objects among the chips such that area and pin constraints are met.

(a) Select "Partition => Automated => Random" This randomly maps the objects to the chips. Note that the area estimates are less than 10,000 for each chip, but the pin estimates exceed their constraints.

(b) Select "Partition => Automated => Group Migration" This applies an algorithm to repartition such that constraints are met, as indicated by a cost of 0.0.

5. **Add a performance constraint**.

(a) Select "Display => User Constraints"

(b) Select the row starting with "XBODY"

(c) Type "300" in the "Value" field. DO NOT press return.

(d) Type "1" in the "Weight" field. DO NOT press return.

(e) Select "Update"

(f) Select "Use" Note in the main display that the estimated execution time for XBODY is 93, which satisfies the constraint of 300.

6. **Create a hardware/software partitioning** to reduce dollar costs.

(a) Select "asic1" in the "Current Allocations" window.

(b) Select "Allocate => Delete" This deletes chip1 from the design. Its objects are unmapped.

(c) Select "Library => Item => Select"

(d) Select "proc8086"

(e) Select OK Note that this processor only costs 20, $ratherthan60$/chip.

(f) Select "Allocate => Add" A processor is added to the design.

(g) Select "Partition => Manual => Unmap all"

(h) Select "Partition => Automated => Random" Note that the performance constraint is being violated.

(i) Select "Partition => Automated => Simulated annealing" Note that the performance constraint is now satisfied.

7. **Create a refined specification** containing the new partitioning information.

(a) Select "Refine => activate" This command is rather slow. The refined specification will be found in 'refined.sc' after completion of the command (you must confirm the filename).

8. **Exit** SpecSyn

(a) Select "File => Quit"

5 Bypassing X windows

The above tutorial used programs running under X windows such as `xspeccharts` and `xspecsyn`. Most of the tools can be run from the command line of UNIX, without requiring X windows. For example, `vol.sc` can be converted to VHDL using `sc2vhdl` as described below. The exception is `xspecsyn`, for which there is no command line version.

Command line tool use is preferred by experienced users who accomplish tasks faster, and who wish to use advanced command line options. For details of the commands and their advanced options, see the man pages for `sc2vhdl, sccheck, scestarea, scestclk, and scestperf`.

The following is an example of SpecCharts translation to VHDL using the `sc2vhdl` command.

(a) Add the `specsyn/bin` directory to your path, as before.

```
% set path=(instdir/specsyn/bin $path)
```

(b) Change directory to the *instdir*`/specsyn/demo` subdirectory.

```
% cd instdir/specsyn/demo
```

(c) You can directly edit any behavior in *VolumeSystem* by modifying the `vol.sc` file using any text editor. Note that the behavior *VolumeSystem* is described as sequential subbehaviors, whereas behavior *Initiate* is described as code.

(d) To translate to VHDL, type:

```
% sc2vhdl vol vol.vhd
```

The `xspeccharts` commands actually executed this command earlier when you selected the translation menu item.

(e) Files similar to those mentioned in Section 2.3 will be created. Simulation can be performed by following the steps of that section.

6 SpecCharts tools

This section includes a brief description of the xspeccharts tool and various other tools for checking and manipulating SpecCharts. Each section is taken directly from a corresponding online man page for the corresponding tool.

6.1 xspeccharts

6.1.1 SYNOPSIS

xspeccharts [speccharts_file]

6.1.2 DESCRIPTION

xspeccharts simplifies the editing/viewing of SpecCharts as well as the application of commands that transform SpecCharts. Although a textual SpecCharts file can be edited using any text editor, xspeccharts supports hierarchical exploration of the file's behaviors. Also, although all SpecCharts related commands can be performed as shell commands, xspeccharts permits application of those commands through selections of menu options.

xspeccharts provides:

- A list of all behaviors. Any behavior in the list can be selected by one click with the left mouse button. The contents of a selected behavior are shown in a viewing area. This area is for rapid viewing only; to edit a behavior, or to view more of it than shown in the viewing area, see the "Edit" button below.
- An "Edit" button, which when pressed, pops up an editor window for the currently selected behavior. Shortcut: two fast clicks in the behavior list, or pressing return in the "Current behavior" field, achieves the same task as pressing the "Edit" button.
  - To edit a subbehavior, select text in the program section of the editor using the mouse, then press "Edit Subbehavior". A new editor window will pop up for the subbehavior.
  - Selecting "Close" in an editor window will close only children of that editor window. If modifications were made in the state or its children, but save has not been pressed, modifications are lost.
  - Selecting "Save" in any editor window will save the entire SpecCharts to the current file. Note: saves can also be performed from the "File" menu; see the MENU BAR section.
- A "Delete" button, which removes the current behavior from the behaviors list.
- A "Hierarchy" button, which uses "xtree" to display the hierarchy of the SpecCharts file as a graphical tree.

6.1.3 MENU OPTIONS

The MENU BAR permits application of several commands:

- "File => New" : Create a new SpecCharts file, removing any existing file from memory. You will be prompted to enter the new filename. Typing Control+N in the xspeccharts window is a shortcut (accelerator) for selecting this menu option.
- "File => Open": Read an existing SpecCharts file into memory, removing any existing file from memory. You will be prompted to select or enter the filename. Accelerator: Control+O.
- "File => Save": Save the current SpecCharts model into the current file. The SpecCharts model need not be syntactically correct. WARNING: if you save a syntactically incorrect SpecCharts model, you won't be able to read that file back into xspeccharts (e.g. after quiting and rerunning xspeccharts later) until all the errors have been fixed in the file using a text editor. Accelerator: Control+S.
- "File => Save As": Save the current SpecCharts model into a file whose name you will be asked to select or enter. The filename becomes the current file. Accelerator: Control+A.
- "File => Check": Perform a syntactic check of the current SpecChart. For a more detailed check, convert to VHDL and apply a VHDL analyzer. See "sccheck".
- "File => Quit": Exits xspeccharts. The current SpecCharts model will be lost unless it was saved to a file. Accelerator: Control+Q.
- "Translators => VHDL (simulatable)": Translates the current SpecCharts model to simulatable VHDL, placing the results in filename.vhd, where filename is the current SpecCharts filename. See "sc2vhdl".
- "Transformations => Flatten": Flattens the selected behavior into sequential statements. See "scflat".

The application defaults file provided with this release must have been merged before running xspeccharts (xrdb -merge INSTALLDIR/speccharts/XSpecsyn).

7 xspecsyn

This section includes a description of the xspecsyn tool for system design. It is taken directly from a corresponding online man page for xspecsyn.

7.1 SYNOPSIS

xspecsyn [-c constraints] [-o objects] [-v] speccharts_file

7.2 DESCRIPTION

Assists a designer in selecting a set of system components, processors, ASICs, and memories, with which to implement a system, and in partitioning the system's functional specification among those components such that constraints are satisfied.

An xspecsyn session typically consists of the following steps:

(a) Functional object selection: The specification's behaviors and variables to be considered for partitioning are selected. (See the "Display => Objects => Choose" menu option).

(b) Allocation: a set of system components are selected and added to the implementation. This usually consists of a few ASICs and possibly a processor. (See the "Partition/Allocate" menu option and the "Allocate => Add" menu option).

(c) Partitioning: the functional objects are assigned to the system components such that constraints are satisfied, either using automated algorithms or manual moves. (See the "Partition" menu option).

(d) Refined specification creation: Once a satisfactory allocation and partition is found, a new specification is created containing the new system components with the functional objects distributed among them, with new behaviors inserted for commmunication and arbitration. (See the "Refine" menu option).

The main display shows the current mapping of functional objects to allocated system components (using indentation), and the current estimates/constraints for each metric applicable to each component or object. If an metric's estimate violates its constraint, an asterisk appears after the estimate/constraint entry.

Below we describe each of xspecsyn's menu options.

- "File => Open" : Opens a new SpecCharts or VHDL specification file. You will be prompted to select or enter a filename. All allocation and partitioning information for the current specification will be lost.
- "File => Quit" : Exit xspecsyn.
- "Display => Objects => Choose" : Selects the granularity at which the partitioning will take place. All variables and behaviors at the top-level of hierarchy must be selected; others are treated as part of their parent unless explicitly selected.

A dialog to appears with a list of all the functional objects from the specification. Objects in the list are indented to show the hierarchy. You can select/deselect any object by clicking on it in the list. Selected objects are highlighted. You can automatically select objects by pressing one of the buttons below the list. For example, to select all array variables, press the "all" button to the right of "Arrays". To select only those array variables at the top-level of hierarchy, press "top" instead. To deselect all array variables, press "none". The highlighting of objects in the list changes to reflect your choice. Any current selection of objects can be saved to a file by pressing the "SaveToFile" button (you will be prompted to enter a file name). Such a file can later be read using the "ReadFromFile" button, or using the "-o objsfile" command line option when starting xspecsyn. It's usually a good idea to save your objects selection.

Once you've selected the objects, press "OK". If your selection does not include all top-level objects, an error message is printed and the selection is ignored.

Note: if your specification consists of sequential subbehaviors at the top-level, the top-level is automatically enclosed into a new behavior called "sc-top". This behavior is needed to activate and deactivate the sequential subbehaviors.

- "Display => Objects => Configure display" : Allows you to inhibit display of certain objects, in order to unclutter the main display. A dialog appears containing a list of all functional objects; displayed objects are shown highlighted. You can select/deselect an object for display by clicking on it. At the top of the list are three special selections. "DISPLAY ALL" selects all objects for display, whereas "DISPLAY NONE" deselects all objects. "DISPLAY ESTS" displays only those objects for which a metric estimate or constraint exists.
- "Display => Columns" : Allows you to select whether a metric column should appear in the main display. A dialog appears with a list of all possible metrics. Those metrics currently being displayed appear highlighted. You can select/deselect a metric by clicking on it. After making all desired changes, press "OK".
- "Display => User constraints" : Allows you to add or modify the constraints on certain design metrics. A list appears where each row represents a constraint recognized by xspecsyn. For example, one row may be:

  Behavior1 avgtime max 0 0

  The first field is the object being constrained, in this case a behavior. The second field is the metric being constrained, in this case the average execution time of the behavior. The third field indicates whether this is a maximum or minimum constraint. The fourth is an integer representing the actual constraint value, and the fifth is an integer which indicates the importance of this constraint relative to the others, with larger numbers indicating more importance.

  To edit a constraint, select the desired row. The fields appear at the bottom of the window. Edit the last two fields with the value and weight. The weight must be 1 or greater for the constraint to be considered, i.e. a weight of 0 causes the constraint to be ignored. After editing these two fields, press "Update". The list of constraints is updated with your changes. After you edit all the constraints you wish to, you can save them to a file by

pressing "Save". Saved constraints can be read later using "ReadFile" or using the "-c constraints" command line option when starting xspecsyn. If you wish to ignore all of the changes you've made, press "Cancel". To use the constraints, press "Use". The non-zero weighted constraints will be incorporated into the cost function.

Constraints associated with system components, such as pins on an ASIC, can not be modified here; you must change the library entry for that component to change such constraints (see "Partition/Allocate – Library => Item => Update").

- "Display => Estimation details" : Allows you to obtain a view of the information which xspecsyn uses to obtain its metric estimates. A dialog appears with a list of metrics. Select the metric for which you want estimation information, and press "OK". A window appears containing text representing the information. The text can be saved to a file by pressing the "Save" button. We shall now describe the information for the metrics of avgtime, area, and pins.

  AVGTIME (performance): avgtime is the average start-to-finish execution time of a behavior. For behaviors with data-dependent execution time, the given branch probabilities are used to determine the average. Execution time is composed of two parts: (1) The communication time, which is time spent accessing other functional objects (variables and behaviors) and waiting for their execution to complete, and (2) The internal computation time, which is the time spent performing computations, assuming communication time is 0.

  An expression is created for each behavior, consisting of internal computation time plus communication times, such as:

```
behavior 'Behavior1':
asictype1__2* | proc8086__31
       + 3 x ( at__x__0|2* + et__x__1 )
       + 1 x ( at__Behavior2__0*|2 + et__Behavior2__5 )
Total time: 16
```

  Since internal computation time is different for different components, the time for each component in the design is shown, with an asterisk indicating which component currently implements the behavior for the current partition. In the above example, the internal computation time is 2 on the asictype1 component, and 31 on the proc8086 component. The asterisk indicates that Behavior1 is currently partitioned to the asictype1 component.

  Communication is shown for each accessed functional object as the sum of two terms: (1) the access time, or the time for signals to propogate between the two objects, (2) the execution time of the accessed object. In the example, Behavior1 accesses x three times for each start-to-finish execution. Each access requires 0 time units if Behavior1 and x are partitioned to the same component, and 2 time units if they are on different components. The asterisk indicates that they are currently on different components. The execution time of x is 1 time unit. For a variable, the execution time is the read or write time of the memory which stores the variable. For a behavior, the execution time is computed as another expression.

The expression is evaluated for current partition and the total shown. In this case the total time is 2 + 3x(2+1) + 1x(0+5) = 16.

AREA (size) : area is the size (e.g. number of transistors) required to implement the functional objects assigned to an ASIC. The area is determined by roughly synthesizing a controller/datapath (cu/dp) architecture to implement the given objects. The area is computed based on several factors: microstates – the number of possible states of the controller. control lines – the number of lines between the cu and dp. statereg bits – the number of bits used to encode the microstates statereg area – the area for a register storing that encoding pla area – the pla area for next-state/control logic comp/mux area – the area of dp components (FUs) and muxes routing area – the area for dp routing The last four areas are summed to give the total. A list of components and muxes and their associated areas is also provided. Below that list is area information about each object, including what components it uses and how many microstates it requires. Also included is information enabling incremental modification of the design when objects are moved; such information will not be useful to you unless you are working on incremental estimation.

PINS : pins is the number of wires which cross the boundary of a component. For each component, the buses which connect an object in this component with an object in another component (or with an external port) are shown, along with their widths. These widths are summed to obtain the total number of pins for the component. Following this information is a textual graph representing the wires needed between all objects. Below that is shown, for each bus, the channels mapped to that bus.

- "Display => Cost function" : Displays details of the cost function used to determine a cost for a given allocation and partition. The cost indicates the "goodness" of a design, with lower cost being better. A cost of 0 is considered ideal.

  The cost function is a sum of all constraint violations. If an estimation for a metric exceeds the metric's constraint, the excess is divided by the constraint (for normalization with other metrics with different units), multiplied by 100, and multiplied by the constraint's weight. The cost is the sum of all such values. Pressing "Save" in the cost function window saves the details to a file.

- "Display => Save to file" : Saves the main display to a file. You will be prompted for a file name. Saving the display to a file is an excellent means for documenting intermediate stages of design.

- "Partition/Allocate => Open window" : Creates a new dialog for partitioning and allocation.

- "Refine => activate" : Activates the refinement tool, which generates a new specification from the old specification and the current allocation and partition. You will be prompted to provide a file name for the new specification.

PARTITION/ALLOCATE dialog description: The dialog consists of two parts. The top half of the dialog shows the current partitioning that is to be done (behaviors to components, variable to memories, or channels to buses) and a list of the current components allocated to the design. The bottom half shows the current library of components, the current library component, the

component class (asic, processor, memory or bus), and characteristics of that component (e.g., maximum area or dollar cost).

- "Libary => Item => Select" : Allows you to select a component from the system component library. Click on the desired component and press OK. The component's characteristics appear on the bottom half of the partition/allocate dialog. If you wish to modify a component's characteristics or add a new component, you can edit the "Current value" field.
- "Library => Item => Add" : Adds a new component, with the name specified in the "Current item" field, to the library.
- "Library => Item => Update" : Updates the current library item to have the currently displayed characteristic values.
- "Library => Item => Delete" : Deletes the current component from the library.
- "Library => Select" : Allows you to select a current library. Note: currently only components from one library can appear in a design. Thus you must select your library before allocating any components.
- "Allocate => Add" : Adds one instance of the current library component to the design. The main display is also updated.
- "Allocate => Update" : Updates the allocated component, whose name you are prompted to enter, so that it is bound to the current library component.
- "Allocate => Delete" : Deletes the allocated component, whose name you are prompted to enter. All objects assigned to that component become unmapped after the component is deleted.
- "Allocate => Delete class" : Allows you to delete all components of a given class (asic, processor, bus, memory). A dialog with each component class currently in the design will appear. Select the class you wish to delete and press "OK".
- "Partition => Manual => Move (text)" : Allows you to move a functional object from one component to another, by typing the names of the object and the destination component. When prompted for the object name, you may use unix-like wildcards to specify more than one object. For example, "*" indicates that all objects are to be moved, while "P*" indicates that all objects whose names begin with the letter P are to be moved. Objects must match
- "Partition => Manual => Move (table)" : Allows you to move a functional object from one component to another, by selecting from a table the object to move and the destination component. When the dialog appears with a list of all objects, you can select one or more objects by clicking on each.
- "Partition => Manual => Group inseparable" : Allows you to combine several objects into a group which cannot be separated during partitioning. When the dialog containing a list of objects appears, you select the objects you want to group, and then enter a name for that group. You may have more than one group. Grouping objects has no effect on estimates or on the refined specification generated; it simply reduces the number of possible partitions.

- "Partition => Manual => Ungroup" : Allows you to ungroup a set of objects which you previously combined into an inseparable group of objects.
- "Partition => Manual => Unmap all" : Removes all functional objects (of the current class) from system components. Those objects are then considered "unmapped", as they were before any partitioning was done.
- "Partition => Automated => Random" : Randomly assigns all unmapped functional objects to system components. Functional objects which were already assigned to a component are not moved.
- "Partition => Automated => Group migration" : Moves objects among components attempting to find a lower-cost partition, using the group migration algorithm, which is an extension of the Kernighan-Lin algorithm with a complexity of n squared.
- "Partition => Automated => Simulated annealing" : Moves objects among components attempting to find a lower-cost partition, using the simulated annealing algorithm.
- "Partition => Automated => Hw/Sw binary" : Moves objects among a processor and ASIC component, attempting to find a lower-cost partition (hardware-software partitioning). Uses the binary-constraint search algorithm, which incorporates simulated annealing.
- "Partition => Hints => Closeness" : Provides a hint as to which object is the best to group with a specific object. The first dialog asks you to select an object. The next dialog asks you specify the relative importance of various closeness criteria. A weight of zero will give that criteria no role in determining closeness. After filling in the weights and pressing "OK", a dialog will appear with a list of objects, ordered by their closeness to the original object. The higher the closeness, the better it is to group.
- "Partition => Hints => Lookahead" : Provides a hint as to which object is the best to move to another component. You are first asked to specify the lookahead distance, which is the size of the sequence of moves you wish lookahead. Normally the distance is 1, since larger distances cause the complexity to increase rapidly. If there are n objects, there are n-to-the-distance moves to be examined.
- "Combined => One per" : Combines allocation and partitioning in a single algorithm. Allocates one of the current library component for each functional object. Usually used to map each channel to its own bus (which is done by default), or sometimes to map each variable to its own memory.
- "Combined => Clique" : Combines allocation and partitioning in a single algorithm. Peforms a clique partitioning.
- "Status => Save" : Saves the current allocation and partition to a file, with the name you provide, which can later be read to restore the design to this status.
- "Status => Restore previous" : Reads an allocation and partition from the specified a file created by an earlier status save. The restore may be performed during a later xspecsyn session than the save.

7.3 NOTES

xspecsyn generates two default files, specsyn_deflib.lib (a default system-component library) and vti100.cmp (a default RT-component set), if they are not present in the directory.

7.4 OPTIONS

- -c constraints : The constraints imposed on the design, such as performance constraints. The constraints file must have been generated during an earlier session of xspecsyn; see "Display => User constraints" for details.
- -o objsfile : The functional objects to be partitioned among system components. The objects file must have been generated during an earlier session of xspecsyn; see "Display => Objects => Choose objects" for details.
- -v : Verbose: causes xspecsyn to print each step it takes to stdout.

8 References

[1] F. Vahid, S. Narayan, and D. Gajski, "Speccharts: A language for system level synthesis," in *Proceedings of the International Symposium on Computer Hardware Description Languages and their Applications*, 1991.

[2] S. Narayan, F. Vahid, and D. Gajski, "System specification and synthesis with the speccharts language," in *Proceedings of the International Conference on Computer-Aided Design*, 1991.

[3] S. Narayan, F. Vahid, and D. Gajski, "Translating system specifications to vhdl," in *Proceedings of the European Conference on Design Automation (EDAC)*, 1991.

[4] S. Narayan, F. Vahid, and D. Gajski, "Modeling with speccharts." UC Irvine, Dept. of ICS, Technical Report 90-20,1990.

[5] F. Vahid and D. Gajski, "Obtaining functionally equivalent simulations using vhdl and a time-shift transformation," in *Proceedings of the International Conference on Computer-Aided Design*, 1991.

[6] S. Narayan and D. Gajski, "System clock estimation based on clock slack minimization," in *Proceedings of the European Design Automation Conference (EuroDAC)*, 1992.

[7] S. Narayan and D. Gajski, "Area and performance estimation from system-level specifications." UC Irvine, Dept. of ICS, Technical Report 92-16,1992.

[8] D. Gajski, F. Vahid, and S. Narayan, "A system-design methodology: Executable-specification refinement," in *Proceedings of the European Conference on Design Automation (EDAC)*, 1994.

A FTP instructions for SpecSyn/SpecCharts

To obtain the release via ftp, you or your company must have already signed a non-disclosure form.

(a) ftp to the address: ftp.ics.uci.edu (128.195.1.1)
login: anonymous
password: your email address

(b) Enter the command: quote site group _____*

(c) Enter the command: quote site gpass _____*

(d) Change directory to: pub/cadlab-releases/sc

(e) Get the file: specsyn3.0b.tar.Z

(* To obtain the passwords for these fields, send email to vahid@ics.uci.edu.)

The file is in tar format and is compressed. If you are not familiar with such files, your system administrator should be able to help. Just a word of warning: the file is about 12Mb compressed, and 25Mb uncompressed, so you'll need lots of disk space, and the transfer may take some time.

You might also be interested in the report pub/cadlab-trs/tr90-20.speccharts_modeling.ps.Z, since it contains several examples along with specific comparisons with Statecharts and with VHDL. Several other SpecCharts papers exist in that directory.

B SpecCharts application notes

This appendix lists a few points concerning the SpecCharts language implementation with this release.

- **Overloading of operators** is not supported.
- **Port modes** are limited to IN, OUT, and INOUT.
- **Nested procedure declarations** are not supported.
- **Symbols added by tools** have `sc_` as a prefix; thus you should not declare symbols with such a prefix.
- **Signal Assignment Statements** – are limited to one waveform in each statement. Thus

  ```
  X <= Y after 5 ns, Z after 10 ns;
  ```

  is not allowed and must be rewritten as

  ```
  X <= Y after 5 ns;
  X <= Z after 10 ns;
  ```

- **Arcs** – an example of a subbehavior with multiple arcs is:

  ```
  SS1 : (TOC, x='1', SS2), (TOC, other, SS3),
        (TI, x='1' and not x'stable, SS2), (TI, timeout(50 ns), SS3);
  SS2 : ...
  SS3 : ...
  ```

- A TOC (transition-on-completion) arc is only evaluated after the subbehavior completes. It is similar to a branch of an 'if' statement. The first arc is traversed if x='1', else the second arc is traversed arc, and so on. The second TOC arc from state SS1 has a special condition "other", which means that if the conditions associated with the other arcs evaluated to false, this "other" arc is traversed. Multiple TOC arcs are checked in the order they appear.

  A TI (transition-immediately) arc causes immediate transition if the expression becomes true. The third arc is a TI with an x rising event, pointing to SS2. The fourth is an EI arc which is transitioned 50 ns after the subbehavior was entered. Valid TI arc expressions are those that are allowed in a VHDL 'wait until' clause (i.e. it must contain at least one signal). If multiple TI arcs should fire simultaneously, they are prioritized in the order they appear.

- **Comments**: both VHDL-style comments (–) and C-style comments (/* */) are allowed; both are stripped during parsing.

  There are two special types of comments in SpecCharts. Three hyphens (—) denote a VHDL-like comment which should not be stripped. These comments will appear when the SpecCharts model is written back out to a file, such as when translating to VHDL or flattening. These comments can only appear where a VHDL declaration or statement can appear.

  Two hyphens followed by a slash (–/) indicate a synthesis directive to be interpreted by SpecSyn. Such directives are not currently written by the user, but rather are generated by SpecSyn itself; in the future, user-specified directives will be incorporated.

- sccheck does not perform an extremely detailed check. For example, it does not check for type mismatches, and does not ensure that a package interface matches its body. It primarily checks for incorrect syntax and for undefined symbols. More detailed checks can be achieved by translating to VHDL and running a VHDL checker.
- Circular definitions, which are allowed in VHDL, are not supported by sc2vhdl when the packages are inlined (the default). The -p option of sc2vhdl can be used to suppress inlining.
- Many of the routines that manipulate SpecCharts do not free memory. Thus, the total memory used by xspeccharts will increase the longer the session is.
- The maximum size of a SpecCharts file is currently 200,000 characters.

C SpecSyn application notes

This appendix lists a few points concerning the SpecSyn tool implementation with this release.

- **A component file** is required for estimation or system design tools. For an example, see `vti100.cmp` file which represents the VLSI Technology Inc. VDP100 library of components. The format for the component file is:

```
CLK   ---
CLASS ---   TYPE ---   DELAY ---   AREA/BIT ---   CONTROL ---   NUM ---
CLASS ---   TYPE ---   DELAY ---   AREA/BIT ---   CONTROL ---   NUM ---
CLASS ---   TYPE ---   DELAY ---   AREA/BIT ---   CONTROL ---   NUM ---
..........
..........
```

  All valid VHDL operators (see Section 7.2.4 of the VHDL LRM) that are used in the design must be specified in the `.cmp` file. There are four classes of components - *operator, register, multiplexer* and *gates*. Currently, only one operator of a given type can be specified. For each component, it is assumed that the area of the component is directly proportional to the number of bits. Thus an *AREA/BIT* parameter must be specified for each component. The `CONTROL` field indicate the number of control lines the component has. `NUM` represents the number of components of a particular type are allocated for implementing the design. The `NUM` parameter for gates, registers and multiplexers is ignored, as is the `CONTROL` parameter for gates.

- Currently, all ASIC components allocated to a design must use the same component file.
- Currently, all system components allocated to a design must be from the same library.
- If you update a library item, but that item is already allocated to the design, the updates do not get reflected in the already allocated items. You should delete those allocations and then reallocate them.
- Sometimes the word 'state' is used in place of 'behavior'. This is an artifact from a previous release in which all behaviors were called states.
- There is no way currently to interrupt or terminate execution of a long-running partitioning algorithm.