# The SpecSynUser's Guide

**Version 2.4**

**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

October 10, 1992

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

# Contents


**1 Software Loading Procedure** 4

**2 The SpecSyn Software File System** 5

2.1 SpecSyn Software Files . . . . . 5

2.2 Naming Requirements for SpecChart Files . . . . . 6

**3 SpecSyn Tutorial** 7

3.1 SpecChart of a Simple Computer . . . . . 7

3.2 SpecSyn Commands using the Graphical Interface . . . . . 9

3.2.1 Starting the XSPECSYN top-level SpecSyn menu . . . . . 9

3.2.2 Viewing/Editing a SpecChart . . . . . 10

3.2.3 Translating a SpecChart to VHDL . . . . . 12

3.2.4 Entering a New SpecChart . . . . . 13

3.2.5 Checking SpecChart Syntax . . . . . 14

3.2.6 Saving a SpecChart . . . . . 15

3.2.7 Estimating Design Clock Cycle . . . . . 15

3.2.8 Estimating Design Area . . . . . 15

3.2.9 Partitioning into Chip Behavioral Descriptions . . . . . 16

3.3 SpecSyn Commands without Graphics . . . . . 17

**4 References** 19

**A SpecChart Application Notes** 20

**B On-line Manual Pages** 22

B.1 sc2dfg . . . . . 22

B.2 sc2vhdl . . . . . 23

B.3 sccheck . . . . . 25
B.4 scestarea . . . . . 26
B.5 scestclk . . . . . 28
B.6 scestperf . . . . . 30
B.7 scs2c . . . . . 32
B.8 scflat . . . . . 33
B.9 scsort . . . . . 34
B.10 sctree . . . . . 35
B.11 xscp . . . . . 36
B.12 xspecsyn . . . . . 40
B.13 xscestarea . . . . . 44
B.14 xscestclk . . . . . 45

# 1 Software Loading Procedure

The SpecSyn software has been tested on SUN-4 platforms with SunOS 4.1.1 . The X-widget based color graphic capture and display tools have been tested under X Windows version X11R5. The SpecSyn software also requires about 20 Mbytes of disk space. The enclosed software release will expire on the date noted on the software release form attached with this document. Complete the following steps to load the SpecSyn software :

1. Insert the SpecSyn Distribution Tape into the tape drive.

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 into the computer's file system 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.

# 2 The SpecSyn Software File System

## 2.1 SpecSyn Software Files

This section briefly describes 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, xscedit, xscestarea, xscestclk, xscestperf` and `xtree` X window graphical 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 executables for the various SpecSyn applications included in this release. Further details can be found in the *man* pages.

   - `sc2dfg` : SpecChart to dataflow-graph translator.
   - `sc2vhdl` : SpecChart to VHDL translator.
   - `sccheck` : A simple SpecChart syntax verifier.
   - `scestarea` : SpecChart area estimator.
   - `scestclk` : SpecChart clock cycle estimator.
   - `scestperf` : SpecChart performance estimator.
   - `scflat` : Flattens SpecChart hierarchy into sequential statements.
   - `scs2c` : Sequential to concurrent behavior conversion.
   - `scsort` : Orders the behaviors within a SpecChart file.
   - `sctree` : Produces a textual tree representing the SpecChart hierarchy.
   - `xdp` : A graph display tool used by the partitioner.
   - `xscestarea` : Graphical interface to the SpecChart area estimator – `scestarea`.
   - `xscestclk` : Graphical interface to the SpecChart clock cycle estimator – `scestclk`.
   - `xscp` : Graphical interface to the SpecChart Behavioral partitioner – `scp`.

- `xspecsyn` : X Windows top-level command menu for working with SpecSyn. Provides the capability to create, edit, or delete SpecChart states, perform syntax checking, converting a SpecChart to VHDL and options for *design partitioning* and *estimation*.
- `xtree` : Graphical display showing the SpecChart hierarchy of behaviors.

4. **demo** : This directory contains a tutorial consisting of a SpecChart for a simple computer, the diagram for which is given in Figure 1.

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

   - `syntax` : SpecChart language syntax.
   - `tokens` : Tokens defined for the SpecChart parser.
   - `user_guide.ps` : Postscript version of this users manual.

6. **man** : This directory consists of the manual pages for the commands `sc2vhdl, sccheck, scestarea, scestclk, scestperf, scflat, scsort, sctree, xscestarea, xscestclk, xscscpart, xspecsyn` and `xtree`. These man pages are also included in this users manual in Appendix B. You can install these man pages in the online manual of your system by updating your MANPATH environmental variable as shown below:

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

   You might have to `rehash` i.e. recompute the internal hash table of the contents of your directories to account for the updated MANPATH. The man pages can then be viewed by using the `man` command. For example, after updating the MANPATH as shown above, the man page for `sc2vhdl` can be viewed by typing:

   ```
   %   man sc2vhdl
   ```

   The man pages can also be viewed directly by using the `nroff` command. For example, to view the man page for `sccheck` from the `specsyn` directory, type :

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

## 2.2 Naming Requirements for SpecChart Files

A SpecChart consists of a hierarchy of behaviors or *states*. The entire SpecChart is stored in a single file. The SpecChart file must have a `.sc` suffix. SpecChart packages, which are similar to VHDL packages, must have a `.scp` suffix.

# 3 SpecSyn Tutorial

This section presents a simple computer example and some of the synthesis tasks that can be performed on it under the **SpecSyn** system-level synthesis framework. The small example demonstrates the `xspecsyn` SpecSyn graphical interface, SpecCharts capture and the `sc2vhdl` , the SpecChart to VHDL translator. Details of the SpecChart language, translation to VHDL, and modeling with the SpecCharts language can be found in [1, 2, 3, 4].

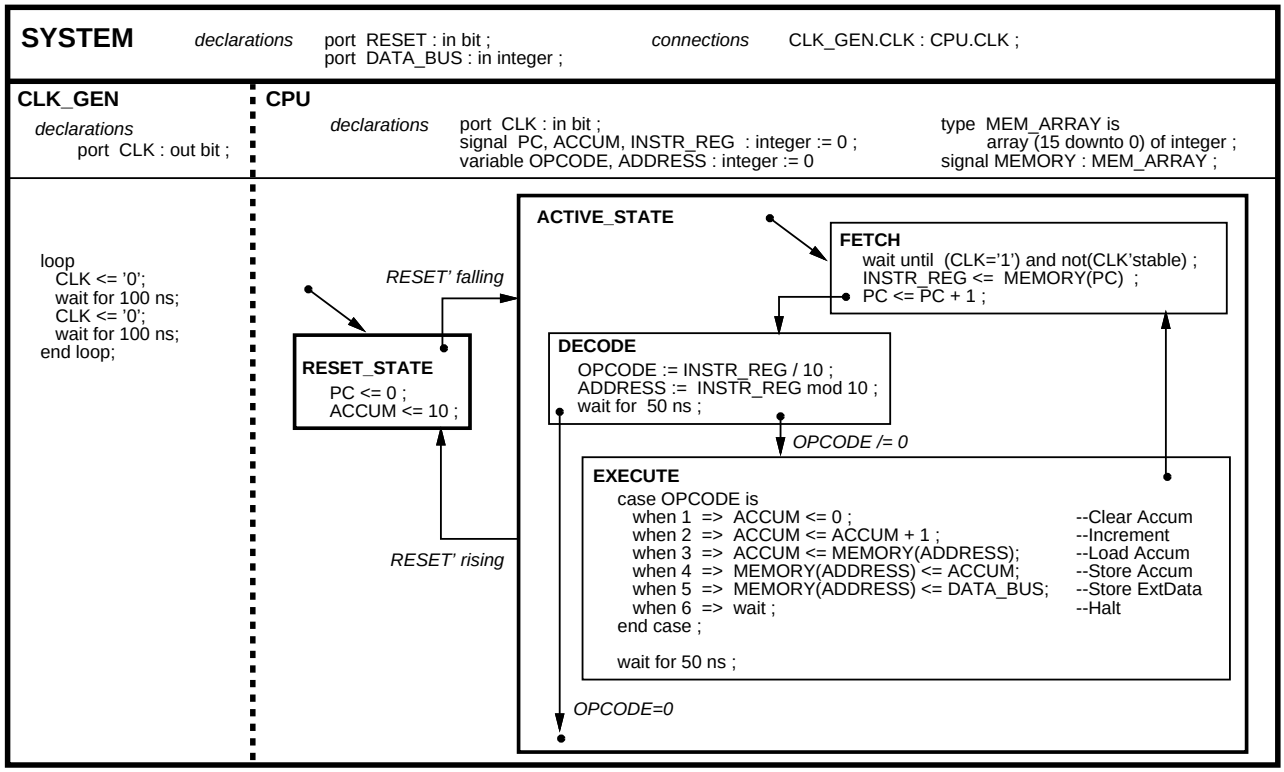


Figure 1: SpecChart of a Simple Computer

## 3.1 SpecChart of a Simple Computer

Figure 1 shows a SpecChart description of a simple computer system, containing a clock generator, a CPU, and a memory. The words behavior and states are used interchangeably in this document.

All the states of the simple computer example, at all levels of the hierarchy, are stored in the file

`system.sc` in the `demo` directory. Each state's functionality can be described either by using VHDL sequential statements (a *leaf* state), *concurrent* substates, or *sequential* substates sequenced by arcs. Each state may contain VHDL declarations whose scope is all descendant states.

RESET and DATA_BUS are declared as external ports of the topmost state SYSTEM. SYSTEM is described using two concurrent substates, CLK_GEN and CPU. CLK_GEN is a *leaf* state which generates a 200 ns clock, described using VHDL sequential statements. The CPU receives the clock from the CLK_GEN via connected ports. CPU contains the memory and three registers declared as signals: PC (program counter), INSTR_REG (instruction register), and ACCUM (accumulator).

CPU is described as two sequential substates, RESET_STATE and ACTIVE_STATE. Initially the CPU is reset in the RESET_STATE. In the RESET_STATE, the memory is initialized and a six instruction program is loaded from location 10. While this is not shown in the figure due to a lack of space, the memory initialization can be seen in state RESET_STATE of file `system.sc`. On the falling edge of the RESET signal, the active state is commenced. The first state is FETCH, which on the rising edge of the clock reads the current memory location into the instruction register. After this in done, DECODE extracts the opcode. EXECUTE then executes the instruction. After execution, FETCH is again commenced.

Also note that two different arcs are used for state transitions. An *exit-immediately* arc (or **EI arc**, drawn originating from the perimeter of the state) is traversed immediately if the arc condition becomes true. The EI arc shown pointing from the ACTIVE_STATE to the RESET_STATE means that regardless of whether the system is currently fetching, decoding, or executing, the CPU is immediately reset on the rising edge of the RESET signal.

An *exit-on-completion* arc (or **EOC arc**, originating from a dot inside a state) is traversed when the state has *completed* AND the arc condition evaluates to true. A leaf state is said to have *completed* when all statements have completed and all signals have received their new value; a state with substates completes when control flows to the *stop* dot (see the *opcode=0* arc in the ACTIVE_STATE).

An EOC arc by default has a condition of true. Note that EOC arcs are found exiting the FETCH, DECODE, EXECUTE and RESET_STATE.

The SpecChart for the computer can be viewed by displaying the file `computer_diag.x` using the "xpic" tool for drawing figures.

```
%    xpic computer_diag.x
```

You can also use a Postscript previewer to view a postscript version of the SpecChart, `computer_diag.ps` . It must be mentioned here that these graphical representations of the SpecChart are separate from our graphical capture of the design. They have been provided only for the sake of convenience (can be used for taking a hard copy of the SpecChart etc.).

## 3.2 SpecSyn Commands using the Graphical Interface

### 3.2.1 Starting the XSPECSYN top-level SpecSyn menu

**xspecsyn** is the top level command menu for working with the SpecSyn system-level synthesis tool. It greatly simplifies the editing/viewing of designs specified with the SpecCharts language as well as the application of synthesis tasks on SpecCharts. Specifically, although a textual SpecChart can be created using any text editor, **xspecsyn** caters to the hierarchical nature of a SpecChart, allowing rapid exploration of behaviors in either a flat or hierarchical manner. Also, although all SpecChart related commands can be issued as shell commands, **xspecsyn** can be used to apply those same commands in a much simpler and more uniform manner. The steps involved in starting the **xspecsyn** top-level SpecSyn menu are:

1. You must start up the X Windows display before proceeding. Add the *instdir*`/specsyn/bin` directory to your path. This will make the SpecSyn software executables accessible from the `demo` subdirectory.

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

2. Change directory to the *instdir*`/specsyn` directory:

   ```
   %   cd instdir/specsyn
   ```

3. The SpecSyn graphics application default file must also be merged with the X Windows server resource database. From the *instdir*`/specsyn` directory, type:

   ```
   %   xrdb -merge XSpecsyn
   ```

4. Change directory to the `demo` subdirectory.

   ```
   %   cd demo
   ```

5. Invoke the top level SpecSyn menu for the simple computer example. The SpecChart for the simple computer is in the file `system.sc`.

   ```
   %   xspecsyn system &
   ```

The `.sc` suffix is not necessary. The **xspecsyn** window pops up. Since the **xspecsyn** command is in the background and produces some textual output, you may have to hit return later to get a prompt.

The **xspecsyn** window is divided into three main parts:

- *Menu Bar*: This consists of pull-down menus which will enable you to do the following:
  - Open, save and syntactically check SpecChart files (see *xspecsyn* man page).
  - Translate SpecCharts to VHDL for simulation (see *sc2vhdl* man page).
  - Flatten the SpecChart hierarchy of states into sequential statements (see *scflat* man page).
  - Invoking estimators for area, clock cycle and performance (see *(x)scestarea, (x)scestclk* and *(x)scestperf* man page).
  - Invoking the behavioral partitioner (see *xscp* man page).
- *SpecChart Capture Command Area*: This provides the designer with the capability to edit/view the behaviors in the SpecChart.
- *SpecChart Viewing Area*: This provides a *rapid* view of the behavior selected in the Capture command area. To edit a behavior, or to view more of it than shown in the viewing area, see the section on editing states below.

### 3.2.2 Viewing/Editing a SpecChart

1. Start the top-level SpecSyn menu, **xspecsyn** as explained in Section 3.2.1 with the simple computer SpecChart in file `system.sc`.

   The *Capture Command area* is on the left-hand side of the **xspecsyn** window and contains the name of the current SpecChart file, `system.sc`. It also contains a list of behaviors in the SpecCharts, in the order read from the SpecChart file. The order in which behaviors occur in the SpecChart file can be changed while saving the SpecChart. The first behavior, SYSTEM, in the list of behaviors is highlighted and is also the current behavior. The declarations and program sections of the current behavior are displayed in the *Viewing area* on the right-hand side of the **xspecsyn** window.

2. **Select "Hierarchy"** by clicking the left mouse button once to view the SpecChart hierarchy graphically in a tree-like structure. The **xtree** window will pop up showing the tree with state SYSTEM at the root. To quit the hierarchical display, type "q" in the **xtree** window.

3. **Select the behavior CLK_GEN** in the behavior list to view the contents of the behavior which will appear in the viewing area on the right hand side. This action also makes CLK_GEN the current behavior. If the viewing area is insufficient to display the entire contents of the behavior, then an "Edit" window must be opened for that behavior, except that no editing will be performed.

4. **Select the behavior SYSTEM** in the behavior list to make SYSTEM the current behavior. Lets say we want to edit the behavior SYSTEM to add a third concurrent sub-behavior, IO_STATE, in addition to CPU and CLK_GEN.

5. **Select the "Edit" option**. This opens an **xscedit** window for the current behavior, SYSTEM, which shows the various sections. We can edit any information in this window by simply placing the cursor at the desired position by clicking the left mouse button. Add IO_STATE as a sub-behavior of SYSTEM in the program section of the **xscedit** window. The **xscedit** window can also be opened by clicking the mouse twice at the behavior (in the behavior list) that is desired to be edited.

6. **Select IO_STATE in the SYSTEM xscedit window** by clicking the left button of the mouse twice in succession. IO_STATE gets highlighted.

7. **Select "Edit Substate"** to open a new **xscedit** window for IO_STATE. We now have two **xscedit** windows – one each for SYSTEM and IO_STATE. In this way we can select any substate of SYSTEM and open and **xscedit** window for it, allowing us to zoom down the hierarchical levels of the SpecChart.

8. Add the VHDL declaration "`signal TEMP : integer ;` " to the declaration section of the **xscedit** window for behavior IO_STATE.

9. **Select "Save"** in the IO_STATE **xscedit** window to save the contents of IO_STATE. A behavior must be saved if a modification is made in it or any of its children. The entire SpecChart `system.sc` is saved when "Save" is selected.

10. **Select "Close"** in the SYSTEM **xscedit** window. This will close the **xscedit** windows of all its children. Thus, even the IO_STATE **xscedit** window is closed automatically since it is a sub-behavior of SYSTEM. In case a **xscedit** window is closed without saving the SpecChart, a warning appears. No warning appeared when we close the SYSTEM **xscedit** window, as we had saved the SpecChart in the previous step. As a result of updating the SpecChart, IO_STATE now appears in the behavior list of the SpecChart.

11. **Select the behavior IO_STATE** in the behavior list to view the contents of the behavior which will appear in the viewing area on the right hand side. This action also makes IO_STATE the current behavior.

12. **Select the "Delete" button** to delete the current behavior IO_STATE from the SpecChart (to avoid changing the SpecChart `system.sc`)

13. Familiarize yourself with the procedure for traversing the hierarchy by zooming down the levels, opening **xscedit** windows for sub-behaviors. When you are done, close all open **xscedit** windows by selecting the "Close" option in each.

14. Before quitting the **xspecsyn** to-level menu, you will have to delete IO_STATE from the program section of SYSTEM. You can achieve this by editing the behavior SYSTEM and saving the SpecChart as explained earlier.

15. **Select File => Quit** (or control-Q) to quit the top-level menu. Any open **xscedit** window will be closed at this time.

### 3.2.3 Translating a SpecChart to VHDL

1. Start the top-level SpecSyn menu, **xspecsyn** as explained in Section 3.2.1 with the simple computer SpecChart in file `system.sc`.

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

   - `system.vhd` which contains a single entity containing the behavior of the entire SpecChart `system.sc`. The VHDL entity generated will have the same name as the root-state of the SpecChart, i.e., SYSTEM. The entity SYSTEM will have port declarations identical to the port declarations declared in the root state of the SpecChart, if any.
   - `system_sim.vhd` which is a skeleton VHDL file, generated only if it does not already exist. This is useful because it instantiates the entity SYSTEM and can be used to provide external stimuli on the ports of entity SYSTEM. It defines a new entity **E** , which can then be simulated. In case a SpecChart being converted to VHDL does not have any external ports in the topmost state or does not require any external stimuli for simulation, the file `system_sim.vhd` can be ignored.

   An additional file `system_sim_demo.vhd` is provided with the release and provides appropriate stimuli to the RESET and DATA_BUS ports of entity SYSTEM.

3. If your installation has the Zycad VHDL Analyzer, compile the VHDL files in the proper order as shown below. Here we shall be using `system_sim_demo.vhd` to provide stimuli to the external

ports of the SpecChart. You could also edit `system_sim.vhd` to add your own stimuli to these ports, and use it instead of `system_sim_demo.vhd` as shown below.

```
%   zvan  system.vhd
%   zvan  system_sim_demo.vhd
```

Use "zvsim" along with the command file `cmds.com` to simulate the VHDL entity, and finally view the simulation results which are logged in the file `run.out` . The script `cmds.com` is specific to the Zycad VHDL Simulator.

```
%   zvsim  -i cmds.com  -t ns  E
%   cat  run.out
```

4. If your installation does not have the Zycad VHDL Analyzer or you wish to use another VHDL analyzer, perform the following steps :

   - Analyze or compile the VHDL file `system.vhd`.
   - Analyze or compile the VHDL stimulus file `system_sim_demo.vhd`.
   - Simulate the entity E of file `system_sim_demo.vhd` for at least 1200 ns.

### 3.2.4 Entering a New SpecChart

In this section, we will enter a SpecChart which will add up all integers from 1 to a specified LIMIT (say, 10).

1. If the **xspecsyn** top-level SpecSyn menu is already open (perhaps as a result of working on another SpecChart), go to the next step. Otherwise, start the **xspecsyn** menu as explained in Section 3.2.1 *without* specifying an input SpecChart file in the command line.

   ```
   %   xscpecsyn &
   ```

2. **Select File => New** from the top-level menu bar. This will prompt you for a file name in which the new SpecChart will be stored. Enter the filename `new.sc` and click "Ok" at the filename prompt.

3. To add a behavior, TOP, to the SpecChart, enter the name of the behavior in the "Current Behavior" window.

4. **Select "Edit"** to add declarations and functionality for behavior TOP.

5. An **xscedit** window for TOP pops up. Add the following to the declaration section for TOP:

```
variable LIMIT : integer ;

variable SUM : integer := 0 ;
```

   Select the *type* of behavior TOP to have "Sequential Substates". Add the following to the program section for TOP:

```
INIT_STATE : (EOC, true, ADD_STATE) ;

ADD_STATE : (EOC, true, stop) ;
```

   Thus we are specifying that the behavior of TOP consists of two sequential sub-behaviors INIT_STATE and ADD_STATE.

6. **Select INIT_STATE, click "Edit Substate"** to open a new **xscedit** window for INIT_STATE. We now have two **xscedit** windows for TOP and INIT_STATE.

7. In the INIT_STATE **xscedit** window, select "Code" to specify that INIT_STATE is a leaf behavior with VHDL code. Add the following to the program section:

```
LIMIT := 10 ;
```

8. **Select "Save" followed by "Close"** in the INIT_STATE **xscedit** window.

9. In a similar manner, open an **xscedit** window for leaf behavior ADD_STATE and add the following to the program section:

```
for INDEX in 1 to LIMIT loop
SUM := SUM + INDEX ;

end loop ;
```

   Save and close the ADD_STATE and TOP **xscedit** windows.

10. The SpecChart with root state TOP will be stored in file `test.sc` and this can be simulated as explained earlier.

### 3.2.5 Checking SpecChart Syntax

Having opened a SpecChart file or entering a new SpecChart, a preliminary syntax verification can be performed 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 exact line in the SpecChart file where the error occurred.

### 3.2.6 Saving a SpecChart

The entire SpecChart file can be saved either when "Save" is selected in an **xscedit** window, or when **File** => **Save** is selected from the top-level **xspecsyn** window.

Before saving a SpecChart, you can specify the order in which the behaviors in the behavior list will be organized in the SpecChart file. You can select either *alphabetical, breadth-first* or *random (or any)* ordering.

Select **Configure** from the top-level menu bar of the **xscpecsyn** window. Click the "Save Order" in the configuration window that pops up. Select the desired save-order and click "Ok". Next time you save the SpecChart, the behaviors will be saved in the order selected. Consequently, when the SpecChart is read in from the file, the behaviors are read in the order *saved.*

### 3.2.7 Estimating Design Clock Cycle

1. Start the top-level SpecSyn menu, **xspecsyn** as explained in Section 3.2.1 with the simple computer SpecChart in file `system.sc`.
2. **Select Estimators** => **Clock** from the **xspecsyn** menu bar to start the SpecSyn clock cycle estimator [6] - **xscestclk**. The **xscestclk** window pops up with the SpecChart file field set to `system.sc`. We will use the default component file, `system.cmp` , to get the functional unit delays.
3. **Select "Maximum Operator Delay"** to estimate the the clock cycle as being the same as the largest operator delay. The frequency of occurrence of operators and their average utilization is displayed by the estimator.
4. **Select "Slack Minimization"** to estimate the the clock cycle using the Slack Minimization method to estimate the clock cycle [6]. The frequency of occurrence of operators and their average utilization is displayed by the estimator.
5. **Select "Quit"** to exit the clock estimator window.

### 3.2.8 Estimating Design Area

1. Start the top-level SpecSyn menu, **xspecsyn** as explained in Section 3.2.1 with the simple computer SpecChart in file `system.sc`.

2. **Select Estimators => Area** from the **xspecsyn** menu bar to start the SpecSyn area estimator - **xscestarea**. The **xscestarea** window pops up with the SpecChart file field set to `system.sc`. We will use the default component file, `system.cmp` , to get the functional unit delays.

3. **Enter CPU** as the "State Name" for which we wish to estimate the area.

4. **Click "Estimate"** to start the estimator for CPU. The estimator will compute and display the components in the design, the microstates in each behavior in the SpecChart, the number of registers, memories, muxes and operators and finally, the area of the control logic, memories and the datapath.

   You may select another state and estimate its area by entering the state name and then clicking "Estimate".

5. **Select "Quit"** to exit the area estimator window.

#### 3.2.9 Partitioning into Chip Behavioral Descriptions

1. **Select Partitioners => Behaviors => Chips** from the **xspecsyn** menu bar to start the SpecSyn behavioral partitioner [7] - **xscp**. The **xscp** window pops up. Open the file "system2.sc" by selecting **File => Open** from the xscp menu bar. We will use the default constraint file, `system2.con`, to get the area, pin, and performance constraints for the partitioner.

2. **Select "SpecToObjs"** and then press the "all" button to the right of the "all-above" label, to select all the top-level behaviors, scalars, and arrays for consideration during partitioning. Press "OK" when done. The program will begin to build the internal information necessary to obtain pin, area, and performance estimates rapidly during partitioning. Views of this information can be viewed by selecting "interconnect", "area", or "performance" in the Estimation Information section.

3. **Select "ObjsToChips"**. This pops up a menu of partitioning options. **Select "Random"**. This applies a random assignment of specification objects to chips to obtain an initial partitioning. Note the Metrics Display and the computed cost value.

4. **Enter 'MEM1' into the "object" field, and 'chip0' into the "dst. chip" field. Then select "Move"**. This manually repartitions by moving MEM1 to chip0.

5. **Select "GroupMigration"**. This applies the group-migration iterative improvement partitioning algorithm, which attempt to find a lower cost partitioning. Note that a zero cost partitioning is found.

6. **Select "File => Refined SpecChart"** from the xscp menu bar. Select "OK" in the file prompt dialog, indicating that the new file should be called 'system2_refined.sc'. This creates a new SpecChart, 'system2_refined.sc', consisting of SYSTEM refined with the partitioning information.

7. **Select "File => Quit"** from the xscp menu bar.

8. **Select "File => Open" in xspecsyn, and then select the file 'system2_refined.sc'**, in order to view the new SpecChart. Note that at the top level there are concurrent behaviors representing chips. You can zoom into each chip to see its constituent behaviors and memories. In this release, interfaces have not yet been fully synthesized between interchip communicating objects.

### 3.3 SpecSyn Commands without Graphics

The tutorial in the previous sections involves using the SpecSyn graphical capture and editing software which require X Windows (Version X11R4). In case problems are experienced with X Window graphics, or if you want to bypass the graphic capture capability of the SpecSyn software, you can perform all the tasks by issuing direct commands from the command line. For example, the SpecChart `system.sc` can be converted to VHDL directly by following the steps given below. For details on these command see the online man pages for *sc2vhdl, sccheck, scestarea, scestclk, and scestperf.*

1. Add the `specsyn/bin` directory to your path. This will make the SpecSyn software executables accessible from the `demo` subdirectory.

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

2. Change directory to the *instdir*`/specsyn/demo` subdirectory.

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

3. You can directly edit any state in the SpecChart by modifying the `system.sc` file using any text editor of your choice. Examples of the three types of states can be found in the following files of the *instdir*`/specsyn/demo` directory:

   - state with concurrent substates : SYSTEM
   - state with sequential substates : ACTIVE_STATE
   - leaf state : EXECUTE

4. To translate the SpecChart to a VHDL file, `system.vhd`, type :

```
%    sc2vhdl system system.vhd
```

5. Files similar to those mentioned in Step 2 of Section 3.2.3 will be created. Simulation can be performed by following the Steps 3 or 4 of that section.

# 4 References


[1] F. Vahid, S. Narayan, and D. Gajski, "SpecCharts: A Language for System Level Synthesis," in *Proc. 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 *Proc. of the International Conference on Computer-Aided Design*, 1991.

[3] S. Narayan, F. Vahid, and D. Gajski, "Translating System Specifications to VHDL," in *Proc. of the European Conference on Design Automation*, 1991.

[4] S. Narayan, F. Vahid, and D. Gajski, "Modeling with SpecCharts." UC Irvine, TR ICS 90-20, 1990.

[5] F. Vahid and D. Gajski, "Obtaining Functionally Equivalent Simulations Using VHDL and a Time-shift Transformation," in *Proc. of the International Conference on Computer-Aided Design*, 1991.

[6] S. Narayan and D. Gajski, "System Clock Estimation based on Clock Slack Minimization," in *Proc. 1st EURO-DAC, Hamburg*, 1992.

[7] F. Vahid and D. Gajski, "Specification Partitioning for System Design," in *Proc. 29th DAC*, 1992.

# A SpecChart Application Notes

This appendix lists a few points concerning the SpecCharts language implementation with this release. Since this is not a full release of the software, a complete updated document on the language is not yet available.

- **Arrays** – (indexed names) are referenced using square brackets, not parenthesis as in VHDL (e.g. $M[x]$, not $M(x)$).
- **Overloading of math operators** is not supported.
- **Port modes** are limited to IN, OUT, and INOUT. We have not tested our simulator on nested procedures. Interface synthesis was not provided with this release; thus channels are not handled. We have not tested designs with INOUT ports *after converting the SpecChart to VHDL.*
- **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;
  ```

- **State names** – All state names must be unique.
- **Aliases** – are not fully supported in translation.
- **Arcs** – an example of a substate with multiple arcs exiting it is given:

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

  An EOC (exit-on-completion) arc is only evaluated after the substate completes. It is like 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 EOC 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 can be traversed. Multiple EOC arcs are checked in the order they appear.

  An EI (exit-immediately) arc causes immediate transition if the expression becomes true. The third arc is an EI with an x rising event, pointing to SS2. The fourth is an EI arc which is

transitioned 50 ns after the substate was entered, assuming it is still active. Valid expressions are those that are allowed in a VHDL 'wait until' clause (i.e. it must contain at least on signal). If multiple EI arcs should fire simultaneously, they are prioritized in the order they appear.

- **Packages** – a limited form is supported. A package is included in a SpecChart by a 'use' declaration. They can only appear in the top state's declarations. (See the syntax file). The package file name must end with a `.scp` extension. If a certain type which is defined in a package is needed by the top level ports, then the "-p" option of the `sc2vhdl` translator must be used when converting to VHDL.

- **Component File**: For every SpecChart, a component file must be specified if the partitioner or estimators are going to be used. The component file contains the clock cycle and the following information for each component type used in the design: class, type, delay, area/bit, control lines, and number of components of that type allocated to the design. In case the component file is not specified using the '-t' option, the area estimator will look for a file called `specchart.cmp` in the current directory. Thus, if a Spec- Chart is stored `system.sc` and no component file is speci- fied, the estimator will look for the file `system.cmp`. If it does not find `system.cmp`, it will be created, having one entry for each of the operator types in VHDL and default values for the fields mentioned above (taken from the VLSI Technology Inc.'s VDP100 datapath library.)

  For an example of a component file, see *instdir*`/specsyn/demo/system.cmp`. The clock cycle in the component file is ignored by the clock estimator. 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 can be 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.

# B On-line Manual Pages

## B.1 sc2dfg

```
sc2dfg(L)                LOCAL COMMANDS                sc2dfg(L)



NAME
     sc2dfg - Outputs the data-flows between  selected  SpecChart
     objects

SYNOPSIS
     sc2dfg specchart_file object_idents dfg_file

DESCRIPTION
     Given a SpecChart and a list of objects, where  each  object
     is  a  state,  procedure,  or  signal/variable  declaration,
     creates a dataflow graph with the objects as vertices.   The
     DFG is output in the form of a VHDL structural netlist.


OPTIONS
BUGS
     Doesn't  declare  user-defined   subtypes   for   the   nets
     representing  data  between  object,  so  applying  a  VHDL
     analyzer will result in undefined symbols.


FILES
     ./<specchart_file>.sc
                         The SpecChart file
     ./<package>.scp     Package files

SEE ALSO
     xspecsyn(L), sc2vhdl(L),

AUTHOR
     Frank Vahid <vahid@ics.uci.edu>, Sanjiv Narayan




U. C. Irvine        Last change: 2 June 1992                  1
```

## B.2 sc2vhdl

```
sc2vhdl(L)              LOCAL COMMANDS              sc2vhdl(L)


NAME
     sc2vhdl - Converts a SpecChart to VHDL

SYNOPSIS
     sc2vhdl [ -t ] [ -v ] specchart_file vhdlfile

DESCRIPTION
     Converts a SpecChart to simulatable VHDL. The VHDL file will
     contain  a single entity implementing the behavior. A skele-
     ton VHDL file (specchart_sim.vhd) is also created, if it did
     not  already  exist, which instantiates the entity as a com-
     ponent, so that you can add stimuli to its ports.


     The transformation steps are printed to stdout as  they  are
     performed.

OPTIONS
     -t   Time shifts the SpecChart before translating.  This  is
          sometimes  necessary  if  the  SpecChart  contains con-
          current behavior that requires  preservation  of  delta
          (micro) time, or EI arcs connecting states with varying
          depths of children which could cause an overdriven sig-
          nal.   This  is  due  to  a  fundamental  limitation of
          translating domain-specific languages to VHDL,  and  is
          not  a  bug  in the translation scheme.  The time-shift
          shifts all time, including delta time, in the  original
          specchart  one  unit, eliminating the above problem but
          causing the simulation output to be  time-shifted  (see
          "Obtaining  Functionally  Equivalent  Simulations Using
          VHDL and a Time-shift Transformation", Frank Vahid,  UC
          Irvine  Technical  Report  #91-33,  April  1991, or the
          paper by the same title published in the  International
          Conference on Computer-Aided Design proceedings, 1991.

     -v   Creates verbose VHDL giving better error messages

     -p   Convert each SpecChart package file to a  VHDL  package
          file,  and  put  all use clauses at the top of the VHDL
          file generated.  Without  this  option,  all  SpecChart
          package  contents will be included directly in the VHDL
          file as declarations.  However, if the  external  ports
          require  a  type  declared  in a package, the -p option
          must be used. This will require analying  each  package
          file created before analyzing the actual VHDL file.
```

FILES

| | |
|---|---|
| ./<filename>.sc | the SpecChart file |
| ./<packages>.scp | the packages |
| ./<specchart>_sim.vhd | a skeleton VHDL file generated, useful for providing stimulus (only created if doesn't already exist) |

BUGS

Interface synthesis is not included with this release. Thus channels/protocols may not be used. Also, connections may only exist between sister substates, not up or down through the hierarchy. Inout ports are not yet supported; use separate ports for in and out.

SEE ALSO

xspecsyn(L), sccheck(L), scflat(L),

AUTHOR

Sanjiv Narayan, Frank Vahid, Computer Science Dept. University of California, Irvine

Comments and questions should be sent to <vahid@ics.uci.edu>.

### B.3 sccheck

```
sccheck(L)              LOCAL COMMANDS              sccheck(L)


NAME
     sccheck - Checks a SpecChart for simple syntax

SYNOPSIS
     sccheck specchart_file

DESCRIPTION
     Parses a SpecChart (top state and all children), performing
     a  simple  check for correct syntax.  Any errors are printed
     on stderr.  To perform a detailed check of a SpecChart, con-
     vert to VHDL and apply a VHDL analyzer.


FILES
     ./<specchart_file>.sc
                         The SpecChart file
     ./<package>.scp     Package files

SEE ALSO
     xspecsyn(L), sc2vhdl(L),

AUTHOR
     Sanjiv Narayan, Frank Vahid <vahid@ics.uci.edu>

U. C. Irvine      Last change: 12 December 1991               1
```

## B.4 scestarea

```
SCESTAREA(L)              LOCAL COMMANDS              SCESTAREA(L)


NAME
     scestarea - estimate area for given SpecChart

SYNOPSIS
     scestarea [ -fvd ] [ -t componentfile ] specchartfile  statename

DESCRIPTION
     scestarea estimates the area of the specified state  in  the
     given SpecChart.

     The component file contains the clock cycle and the  follow-
     ing  information for each component type used in the design:
     class, type, delay, area/bit, control lines, and  number  of
     components  of  that  type allocated to the design.  In case
     the component  file is not specified using the '-t'  option,
     the   area   estimator   will   look   for   a  file  called
     'specchart.cmp' in the current directory. Thus, if  a  Spec-
     Chart is stored  'system.sc' and no component file is speci-
     fied, the estimator will look for the file 'system.cmp'.  If
     it  does  not  find 'system.cmp', it will be created, having
     one entry for each of the operator types in VHDL and default
     values for the fields mentioned above.

     If no options are specified, the estimator by  default  pro-
     duces a single-line non-verbose output.

OPTIONS
     -t compfile
          File containing the component information. The  default
          file is 'specchart.cmp'.

     -d   Output  the  results  on  three  files  :   ",tmpcomp",
          ",tmpdata",  and  ",tmpstate" which contain the results
          of the estimation for use by xscestarea.

     -f   Estimate the area by  first  building  a  special  data
          structure  containing  various  pieces  of  information
          about the SpecChart, and then using it to  compute  the
          area. Not observable speedups  if this option is used.

     -v   Verbose output which also lists the  number  of  micro-
          states and components.

FILES
     ./*.sc              SpecChart files.
     ./*.scp             SpecChart packages.
```

```
     ./*.cmp             File containing component information.
     ./,tmpcomp          File containing the list  of  components
                         when the ‘-d’ flag is used.
     ./,tmpdata          File containing the data such as  number
                         of  bits in state register etc. when the
                         ‘-d’ flag is used.
     ./,tmpstate         File containing  the  number  of  micro-
                         states  required  by  each leaf behavior
                         when the ‘-d’ flag is used.

SEE ALSO
     xscestarea

AUTHOR
     Sanjiv Narayan  (narayan@ics.uci.edu)
```

## B.5 scestclk

```
SCESTCLK(L)              LOCAL COMMANDS              SCESTCLK(L)


NAME
     scestclk - estimate clock cycle for given SpecChart

SYNOPSIS
     scestclk [ -wmb ] [ -vd ] [ -t componentfile ] specchart


DESCRIPTION
     scestclk estimates the clock cycle for a given SpecChart and
     an associated component file.

     The component file contains the following information for
     each component type used in the design: class, type, delay,
     area/bit, control lines, and number of components of that
     type allocated to the design. In case the component file
     is not specified using the '-t' option, the estimator will
     look for a file called 'specchart.cmp' in the current direc-
     tory. Thus, if a SpecChart is stored 'system.sc' and no
     component file is specified, the estimator will look for the
     file 'system.cmp'. If it does not find 'system.cmp', it will
     be created, having one entry for each of the operator types
     in VHDL and default values for the fields mentioned above.
     (Note: For the clock cycle estimator, the CLK specified in
     the component file is ignored).

     The clock cycle estimator can estimate the clock using
     either the Clock Slack Minimization method or the Maximum
     Operato Delay method. In the former, the clock cycle is
     computed so as to minimize the slack involved when a par-
     ticular operator is used in a control step. In the latter
     method, the largest delay of any operator is computed as the
     clock cycle.

     If no options are specified, the estimator by default pro-
     duces a non-verbose output using the Clock Slack Minimizati-
     on method.

OPTIONS
     -w   Estimate the clock cycle using the Slack Minimization
          method

     -m   Estimate the clock cycle using the Maximum Operator
          Delay method

     -b   Estimate the clock cycle using both the Slack Minimi-
          zation and the Maximum Operator Delay methods
```

-v Verbose output which lists the number of occurrences of each operator and operator delays.

-d Output the results on three files : ",tmpclkest" which contains the clock cycle estimated, "tmpclkutil" which contains the utilization of the operators for that clock cycle, and ",tmpclkusage" which contains the number of operators of each type and their delays.

-t componentfile
File containing the component information. The default file is ‘specchart.cmp’.

FILES

| | |
|---|---|
| ./*.sc | SpecChart files. |
| ./*.scp | SpecChart packages. |
| ./*.cmp | File containing component information. |
| ,tmpclkutil | File containing the clock cycle utilization when the ‘-d’ flag is used. |
| ,tmpclkest | File containing the clock cycle estimated when the ‘-d’ flag is used. |
| ,tmpclkusage | File containing the number of operators of each type in the design and their delays and frequency of occurrence, created when th e‘-d’ flag is used. |

SEE ALSO

xscestclk,

AUTHOR

Sanjiv Narayan (narayan@ics.uci.edu)

## B.6 scestperf

SCESTPERF(L) LOCAL COMMANDS SCESTPERF(L)

NAME

scestperf - estimates performance for a given SpecChart state or procedure.

SYNOPSIS

scestperf [ -t compfile ] [ -dipv ] specchart statename

DESCRIPTION

Computes the execution time of the specified procedure or state in terms of clock cycles. For a non-leaf state, the estimator performs control-flow analysis of the state sequencing at each hierarchical level. The state being estimated must not have any descendant concurrent substates. Each non-leaf state MUST have "stop" dot in it.

The component file contains the clock cycle and the following information for each component type used in the design: class, type, delay, area/bit, control lines, and number of components of that type allocated to the design. In case the component file is not specified using the '-t' option, the estimator will look for a file called 'specchart.cmp' in the current directory. Thus, if a SpecChart is stored 'system.sc' and no component file is specified, the estimator will look for the file 'system.cmp'. If it does not find 'system.cmp', it will be created, having one entry for each of the operator types in VHDL and default values for the fields mentioned above.

If no options are specified, the estimator by default produces a non-verbose output without inlining procedures and using the component file 'specchart.cmp'.

OPTIONS

-t File containing the component information. The default file is 'specchart.cmp'.

-d Dumps results to files, for use by xscestperf.

-i Inline procedures. If this flag is omitted, the estimator will ignore all procedure calls encountered in the VHDL code.

-p Indicates that the "state_name" parameter is actually a procedure declared in the SpecChart. The performance of the procedure is then estimated.

```
     -v   Verbose mode, prints  estimates  at  each  hierarchical
          level.

FILES
     ./*.sc     SpecChart files
     ./*.scp    Packages required by the specchart.
     ./*.cmp    File containing component information.

BUGS
     Currently, channels are treated as procedure calls.


SEE ALSO
     xspecsyn(L), xscest(L), xscestperf(L)

AUTHOR
     Sanjiv Narayan (narayan@ics.uci.edu), Computer Science Dept,
     University of California, Irvine
```

## B.7 scs2c

```
scs2c(L)                LOCAL COMMANDS                  scs2c(L)



NAME
     scs2c - Convertes a SpecChart state or procedure to  a  pro-
     cess

SYNOPSIS
     scs2c specchart_file behavior_ident out_specchart_file

DESCRIPTION
     Given a SpecChart and a behavior identifier, creates  a  new
     specchart  where the behavior is a process (a top-level con-
     current substate).


OPTIONS
BUGS
     Only works for simple cases.


FILES
     ./<specchart_file>.sc
                         The SpecChart file
     ./<package>.scp     Package files

SEE ALSO
     xspecsyn(L),

AUTHOR
     Frank Vahid <vahid@ics.uci.edu>, Sanjiv Narayan
```

## B.8 scflat

```
\subsection{scflat}
\begin{verbatim}
scflat(L)                LOCAL COMMANDS                scflat(L)



NAME
     scflat - Flattens a SpecChart  state's  contents  from  sub-
     states to code

SYNOPSIS
     scflat [ -s ] specchart_file statename out_sc_file

DESCRIPTION
     Converts a state with sequential  substates  into  one  with
     only code (sequential statements).  The code is comprised of
     a loop, a state variable, and a case statement,  where  each
     case  branch is the contents of a leaf state.  Such flatten-
     ing is useful as a step in creating input for  a  high-level
     synthesis  tool.   The code would generally have been diffi-
     cult to write by hand.


OPTIONS
     -s   Flatten to the point of leaf states sequenced by  arcs.
          Hierarchy  is  thus removed, leaving a state-transition
          diagram with code in each state.



FILES
     ./<specchart_file>.sc
                         The SpecChart file
     ./<package>.scp     Package files

SEE ALSO
     xspecsyn(L), sc2vhdl(L), sctree(L),

AUTHOR
     Sanjiv Narayan, Frank Vahid <vahid@ics.uci.edu>




U. C. Irvine        Last change: 2 June 1992                   1
```

### B.9 scsort

```
scsort(L)                LOCAL COMMANDS                scsort(L)



NAME
     scsort - Sorts the order of appearance of  states  within  a
     SpecChart file

SYNOPSIS
     scsort specchart_file out_sc_file sort_style


DESCRIPTION
     Writes the input SpecChart  into  the  output  file  in  the
     sorted  order  specified.  Valid sort orders are: 0 (any), 1
     (alphabetical), and 2 (breadth-first).


FILES
     ./<specchart_file>.sc
                         The SpecChart file
     ./<package>.scp     Package files

SEE ALSO
     xspecsyn(L),

AUTHOR
     Sanjiv Narayan, Frank Vahid <vahid@ics.uci.edu>




U. C. Irvine        Last change: 2 June 1992                   1
```

## B.10 sctree

```
sctree(L)                LOCAL COMMANDS                sctree(L)


NAME
     sctree - Outputs a textual tree representing  a  SpecChart's
     hierarchy

SYNOPSIS
     sctree specchart_file treefile


DESCRIPTION
     Writes the input SpecChart as a textual tree, where each arc
     is  between  a parent and child (i.e. state and substate) in
     the SpecChart hierarchy. Useful as input to xtree.


FILES
     ./<specchart_file>.sc
                         The SpecChart file
     ./<package>.scp     Package files

SEE ALSO
     xspecsyn(L), xtree(L),

AUTHOR
     Sanjiv Narayan, Frank Vahid <vahid@ics.uci.edu>

U. C. Irvine        Last change: 2 June 1992                  1
```

## B.11 xscp

```
xscp(L)                    LOCAL COMMANDS                    xscp(L)


NAME
     xscp - The SpecSyn behavioral partitioning tool  with  an  X
     interface

SYNOPSIS
     xscp [ -c constraints ] [ -t components ] [ -o objects  ]  [
     -v ] [ -r ] specchart_file

DESCRIPTION
     Creates an X application to assist in partitioning  a  Spec-
     Chart  behavioral description into multiple chips, such that
     a set of area, pin, and performance constraints  are  satis-
     fied.

     A partitioning session typically consists of  the  following
     steps:

     1. Object  selection:  The SpecChart behaviors, memories
     (composite  declarations  except bit_vectors), and registers
     (scalar declarations and bit_vectors) to be  considered  for
     partitioning are selected. Since behaviors are hierarchical,
     unselected behaviors are subsumed by  the  closest  selected
     ancestor  behavior.  A  valid  selection must "cover" the
     entire SpecChart, i.e. all  registers,  memories,  and
     behaviors  must  be selected either explicitly or implicitly
     by being subsumed by an ancestor behavior.

     Selection can be achieved by applying the "SpecToObjs"  but-
     ton.  A  list  of  all possible objects in the SpecChart is
     displayed, from which  objects  may  be  selected.  The  -o
     option allows objects to initially be read in from a file.

     2. Estimation information creation: The  partitioning  algo-
     rithms  require  estimations  of pins, area, and performance
     for a given partition.  Estimation  information  is  obtained
     as  soon  as  objects are selected. For pins, estimates are
     obtained by creating a hypergraph where the selected objects
     serve  as vertices and edges represent wires needed for com-
     munication. For area, the datapath components  and  micro-
     states are estimated for each object; to estimate area for a
     given chip with some objects, the  components  are  combined
     into a datapath, the microstates into a controller, and wir-
     ing is estimated.  Area estimation is done quickly by incre-
     mentally  changing  these  estimates  during  partitioning,
     rather than reestimating entirely.  For performance,  for
     each  object  an average execution-time equation is built as
     the sum of internal computation time and communication time,
```

the latter consisting of accesses to other objects. Portions of each of these three forms of estimation information can be viewed in xscp by clicking the "interconnect", "area", or "performance" buttons. In the future, one should be able to manually modify this information.

3. Partitioning objects among chips: Several algorithms can be applied to create an initial complete partitioning of objects into chips. "Cluster1": uses hierarchical clustering. "Random": randomly assigns each object to a partition. "OnePerChip" assigns each object to its own chip; this is most useful for examing the contribution of each object to chip pin and area estimates.

Once an initial partition is obtained, an iterative improvement algorithm can be applied : "GroupMigration", which is a variation of Kernighan/Lin. Manual partitioning can also be applied at any time by entering the object name and the destination chip name and pressing "Move".

For any given partitioning, area, pin, and performance metrics are shown in the Metrics Display, along with the current partitioning and the result of applying the cost function. Details of each of these metrics may be examined by pressing the appropriate button at the bottom of the window ("GetPinInfo", "GetAreaInfo", "GetPerfInfo", "GetCostInfo".

4. Refined SpecChart Creation: Once a satisfactory partitioning is found, a new SpecChart can be created containing the chip partitioning information, by selecting the menu item "File --> Refined Specchart". This release does not fully synthesize the communication interfaces.

In addition to the above, one may group certain objects together so that partitioning will always place all objects in a group on the same chip ("ObjsToGroups"). One may also group storage objects into memories, meaning that not only can't those objects be separated, but that communication is estimated over a single bus.

Constraints are used by the cost function to evaluate a given partitioning. They can be modified by selecting "Modify", which creates a list of all possible constraints for the given SpecChart. Any constraint with a weight of 0 is ignored.

OPTIONS

-c constraints

Can be used to specify initial constraints to the partitioner, found in the file "constraints". If this

option is not specified, xscp by default searches for the file <specchart>.con. If not found, a default <specchart>.con file is created and used. The created file will exist even after xscp has been exited. The easiest way to create a constraints file is to let the default be created and then to use "Modify" in xscp.

-t components
Specifies the file containing the component and clock information to be used for estimation purposes. See "scestarea".

-o objsfile
The objects in objsfile serve as the initially selected objects to be partitioned. This is useful if many objects would otherwise need to be selected manually into the objects selection section after starting xscp.

-v Verbose mode: more detailed messages are printed during execution of commands.

FILES

./<filename>.sc the SpecChart file
./<packages>.scp the packages
./<filename>.con the partitioning constraints
./<filename>.cmp the components file used for estimations

Being a prototype tool, there are a few. Doesn't handle objects that are procedures/functions and are called by concurrent behaviors; such routines should not be selected as partitioning objects. This release does not fully synthesize the communication interfaces in the refined SpecChart.

SEE ALSO

xspecsyn(L), xscestarea(L), xscestperf(L),

AUTHOR

Frank Vahid, Sanjiv Narayan Computer Science Dept. University of California, Irvine

Comments and questions should be sent to <vahid@ics.uci.edu>.

## B.12 xspecsyn

```
xspecsyn(L)              LOCAL COMMANDS              xspecsyn(L)


NAME
     xspecsyn - X11 graphical program for working with SpecCharts

SYNOPSIS
     xspecsyn [ specchart_file ]

DESCRIPTION
     This is the top level command menu for working with SpecSyn,
     a system-level synthesis tool. It greatly simplifies the
     editing/viewing of models captured in the SpecCharts
     language as well as the application of commands on Spec-
     Charts. Specifically, although a textual SpecChart can be
     created using any text editor, xspecsyn caters to the
     hierarchical nature of a SpecChart, allowing rapid explora-
     tion of behaviors in either a flat or hierarchical manner.
     Also, although all SpecChart related commands can be per-
     formed as shell commands, xspecsyn can be used to apply
     those same commands in a much simpler and more uniform
     manner.  xspecsyn also permits easy startup of synthesis
     commands, such as estimation and partitioning.

     xspecsyn consists of three areas: a menu bar, a SpecChart-
     capture command area, and a SpecChart-behavior viewing area.

     The CAPTURE command area and the VIEWING area provides:


     * A list of all states in the SpecChart. Any state in the
     list can be selected by one click with the left mouse but-
     ton. The contents of a selected state are shown in the
     viewing area. This area is for rapid viewing only; to edit
     a state, 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 state. Note: two fast
     clicks in the state list, or hitting return in the "Current
     behavior" field, achieves the same task as pressing the
     "Edit" button.


     -- To edit a substate, select text in the program section of
     the editor using the mouse, then press "Edit Substate". A
     new editor window will pop up for the substate.

     -- 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 the SpecChart has not been saved to a file, a warning appears.

-- Selecting "Save" in any editor window will save the entire SpecChart 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 state from the states list.

* A "Hierarchy" button, which uses "xtree" to display the hierarchy of the SpecChart as a graphical tree.

The MENU BAR

File --> New : Create a new SpecChart file, removing any existing SpecChart from memory. You will be prompted to enter the new filename.

File --> Open: Read an existing SpecChart file into memory, removing any existing SpecChart from memory. You will be prompted to select or enter the filename. Pressing '^O' (Control+O) anywhere in the xspecsyn window accomplishes the same task.

File --> Save: Save the current SpecChart into the current file. The SpecChart need not be syntactically correct. Pressing '^S' (Control+S) anywhere in the xspecsyn window accomplishes the same task.

File --> Save As: Save the current SpecChart into a file whose name you will be asked to select or enter. The filename becomes the current file. Pressing '^A' (Control+A) anywhere in the xspecsyn window accomplishes the same task.

File --> Check: Perform a general syntactic check of the current SpecChart. For a detailed check, convert to VHDL and apply a VHDL analyzer. This selection actually writes the current SpecChart to a temporary file, and then applies the "sccheck" command.

File --> Quit: Exits xspecsyn. The current SpecChart will be lost unless it was saved to a file. Pressing '^Q' (Control+Q) anywhere in the xspecsyn window accomplishes the same task.

Translators --> VHDL (simulatable): Translates the current SpecChart to simulatable VHDL, placing the results in filename.vhd, where filename is the current SpecChart filename. See "sc2vhdl".

Translators --> Dataflow-graph: Given a set of object names (where each object is a state, procedure, scalar or array), outputs a VHDL structural netlist indicating the dataflows between the objects. See "sc2dfg".

Transformations --> Enums2Const: converts enumerated types to constants.

Transformations --> Flatten: Flattens a given state in the SpecChart (as selected in the state list) into sequential statements. This is most commonly used to flatten sequential substates into a case statement. See "scflat".

Transformations --> Inline: Inlines a specified procedure. See "scinline".

Transformations --> To Process: Converts the given state or procedure to a concurrent process. See "scs2c".

Transformations --> Unique Symbols: Makes all symbols in the SpecChart unique.

Estimators --> Area: Fires off the SpecChart area estimator graphical interface.

Estimators --> Clock: Fires off the SpecChart clock estimator graphical interface.

Estimators --> Performance: executes the command "scestperf" with the state name parameter being the currently selected state.

Configure : Currently, the only configuration choice is the order in which states are saved in a file. The choices are: any, alphabetical, and breadth-first. The latter is considerably slower than the first two, and requires a syntactically correct SpecChart.

The application defaults file provided with this release must have been merged before running xspecsyn (xrdb -merge <INSTALLDIR>/speccharts/app-defaults).

OPTIONS

specchart_file

The name of an initial specchart file

FILES

./<specchart_file>.sc

the SpecChart file

SEE ALSO
     sccheck(L),    sc2vhdl(L),     scflat(L),     xscestarea(L),
     xscestclk(L), scsort(L),

AUTHOR
     Frank Vahid <vahid@ics.uci.edu>

## B.13 xscestarea

```
XSCESTAREA(L)            LOCAL COMMANDS            XSCESTAREA(L)


NAME
     xscestarea - graphical interface to the SpecSyn area estima-
     tor, "scetsarea"

SYNOPSIS
     xscestarea specchart_file [ statename ]

DESCRIPTION
     xscestarea provides a graphical interface to the "scestarea"
     area  estimator,  which  estimates the area of the specified
     state in the given SpecChart.

     If the name of the state  for  which  the  area  has  to  be
     estimated  is  not specified in the command line, it must be
     entered in the "State Name" field of the xscestarea window.

     A component file needs to be specified  which  contains  the
     clock cycle and the following information for each component
     type used in the design: class, type, delay, area/bit,  con-
     trol  lines, and number of components of that type allocated
     to the design.  In case the component file is not  specified
     in the "Component File"  field of the xscestarea window, the
     area  estimator  will  look  for  a  default   file   called
     'specchart.cmp' in the current directory.

FILES
     ./*.sc              SpecChart files.
     ./*.scp             SpecChart packages.
     ./*.cmp             File containing component information.

SEE ALSO
     scestarea

AUTHOR
     Sanjiv Narayan  (narayan@ics.uci.edu)




U. C. Irvine        Last change: 4 June 1992                  1
```

## B.14 xscestclk

```
XSCESTCLK(L)              LOCAL COMMANDS              XSCESTCLK(L)


NAME
     xscestclk - graphical interface to the  SpecSyn Clock  esti-
     mator - "scestclk"

SYNOPSIS
     xscestclk spechart_file


DESCRIPTION
     xscestclk is a graphical interface to the "scestclk" estima-
     tor which estimates the clock  cycle,  given a SpecChart and
     an associated component file.

     A component file needs to be specified  which  contains  the
     clock cycle and the following information for each component
     type used in the design: class, type, delay, area/bit,  con-
     trol  lines, and number of components of that type allocated
     to the design.  In case the component file is not  specified
     in the "Component File"  field of the xscestarea window, the
     area  estimator  will  look  for  a  default   file   called
     'specchart.cmp' in the current directory.

     The clock cycle  estimator  can  estimate  the  clock  using
     either  the Clock Slack   Minimization method or the Maximum
     Operato Delay method.  In the former,  the  clock  cycle  is
     computed so as to minimize the slack    involved when a par-
     ticular operator is used in a control  step.  In  the  latter
     method, the largest delay of any operator is computed as the
     clock cycle.

FILES
     ./*.sc              SpecChart files.
     ./*.scp             SpecChart packages.
     ./*.cmp             File containing component information

SEE ALSO
     scestclk

AUTHOR
     Sanjiv Narayan  (narayan@ics.uci.edu)




U. C. Irvine        Last change: 4 June 1992                    1
```