SPACE TUTORIAL

N.P. van der Meijs, A.J. van Genderen

Circuits and Systems Group
Department of Electrical Engineering
Delft University of Technology
The Netherlands

Report ET-NT 92.22

Last revision: May, 2003.

1. Introduction

This tutorial describes how to use the *space* layout-to-circuit extractor. *Space* is used to extract a netlist from a given mask-level layout. Depending on the options specified when invoking *space*, the netlist

- can be flat, hierarchical (the same hierarchical structure as present in the layout) or mixed flat/hierarchical.
- can contain interconnect capacitances (e.g. capacitances to the substrate, fringe capacitances, cross-over coupling capacitances and capacitances between parallel wires)
- can contain interconnect and substrate resistances.

The extracted network (including its parasitic information) can be used to verify the functionality and the performance of the circuit, e.g. by means of *post-layout* simulation or timing analysis.

This document is structured as follows. Section 2 gives an introduction to some basic concepts that are used in connection with the extraction program. It gives some definitions, it describes how the database that stores the design data is arranged, and it shows the relation of the extraction tools with the Unix file system and the Unix command shell. Section 3 gives an example of a layout-to-circuit extraction for a particular circuit. This section discusses each of the different tools that are used in conjunction with the extraction program and it shows the relation between these tools by calling them for the given extraction example. Finally, the last sections describe some extraction strategies for particular applications such as special device extraction and sea-of-gates circuit extraction.

> **NOTE:**
> The Space System, the *space* extractor and its tools, is based on the Release 3 of the Nelsis IC Design System.

> **NOTE:**
> Throughout this document, we assume that all installation-related space files and directories are stored under the **/usr/cacd** directory. If this ICDPATH directory is different at your site, please read these phrases with the correct pathname substituted for **/usr/cacd**. Consult your system administrator if you don't know the correct pathname.
>
> If you are going to use *space*, make sure that there is an entry **/usr/cacd/share/bin** in the command search path of your Unix shell. Note that the environment variable ICDPATH does not need to be set manually, it is set by a shell script before each tool invocation.

2. Basic Concepts

2.1 Projects

Space operates in a database that is called a *project*. In this project, it finds the layout data and stores the extracted circuit data. Under Unix, a project is a directory with two subdirectories (called *view directories*, layout and circuit) where the actual data is stored, and a *projlist* file. The *projlist* file contains a list of related projects (e.g. *library projects*), as is discussed in section 2.2. All design data that belong to a project are stored in the file structure underneath the project directory. If necessary, project directories can be moved, copied or archived like ordinary Unix directories. Note that if a project is referenced, it is a library project. Because other projects are using it and know where it is located, it is not wise to move this projects to another location.

Each layout is designed in a particular process (e.g. ES2 ECPD07 or MOSIS scalable CMOS) and using a particular value of λ (lambda)). The process and λ value are set for a particular project; all cells in the project share the same process and λ value. This information has to be specified when the project is created, and is, among other information, stored in file named *.dmrc* in the project directory.

Lambda is a specification of the elementary distance unit of the layout data. All physical dimensions of the design descriptions are expressed in (integer) values of this λ unit. Lambda should be specified small enough in order to represent all valid layout positions. Also, it should be specified not too small in order to prevent integer overflows for large layouts.

The technology files for *space* contain capacitance values per meter or per meter^2. *Space* uses the value of λ to convert a layout in the project database into its (would be) physical dimensions, to produce its (would be) capacitances and/or resistances. For example, with λ = 0.5 microns a layout feature of 4 λ units becomes 2 microns when the design is actually extracted. Lambda is constant for all cells in a project, but different projects can have a different λ. For current processes, typical values of λ are around 0.01-0.1 micron or $10^{-8} - 10^{-7}$ m.

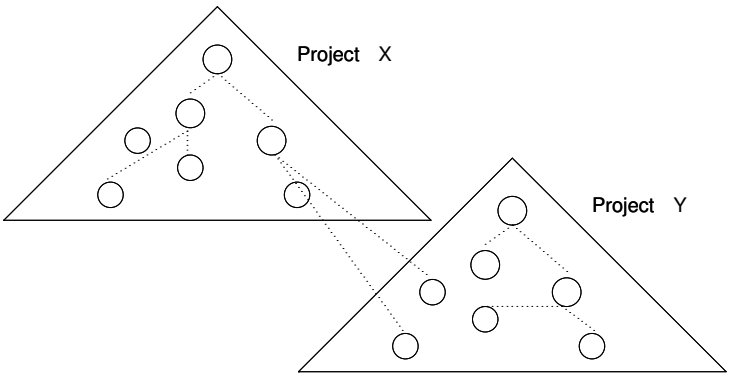
2.2 Library Projects

Projects form local databases. There is no direct interference, such as name clashes, between data in different projects. However, projects are not completely isolated from each other. An *import mechanism* permits *remote* cells (cells in other projects) to be used locally, without actually copying them. A cell from another project may be imported into the local project to make it available for *instantation* (hierarchical inclusion), while an *alias* (local name) may be assigned to it.

This concept is particularly useful if remote projects contain large collections of reusable predefined cells. These projects can then operate as *library projects*. The cells in a library project can be accessed from other projects (with the same technology and λ), without copying of design data.

Such a facility is virtually mandatory for certain design styles such as standard-cell or gate-array design, but is also very useful for full-custom design.

An example of multiple projects with imported cells is depicted in the figure below.



As one can see, some of the cells in projects X and Y are defined hierarchically, instantiating local cells. Project X however also contains cells which instantiate cells from project Y. These remote cells have been imported previously into project X, before instantiating (= referring) them within this project.

2.3 Technology Independence

Space, and its auxiliary programs, are technology independent. Information about the masks present, elements to be recognized, values of parasitics etc., is stored in certain files in a process directory. For each process supported, there is a directory that in turn contains the tool-specific data files. These directories together are called the *technology base*, and reside in a special place, known by all tools, in the Unix directory tree.

Introducing a new process merely implies that a new directory with the corresponding data files has to be created. More information on how to enter a new process, the format of the datafiles etc., is given elsewhere.

2.4 Tools

Space is accompanied by a number of auxiliary tools that perform tasks like creating and removing projects, listing the contents of a projects, extracting a SPICE netlist from a project and so on.

The tools are normally invoked by the user through the Unix command interpreter, or shell. That is, the user works from a Unix shell and all corresponding mechanisms apply to the process of tool invocation. For instance, the user may go through the file system and position himself in a (project) directory with such commands as *ls* and *cd*, use *ps* to see which tools are running, and store the output of a tool in a particular file with the

usual IO-redirect mechanisms. Also, the selection of the executable that will be run is done using the usual search path mechanisms.

As an example, we show how one may move to a project ‘‘alu_proj’’ and invoke a tool to inspect the contents of that project. With the typical Unix phrase

```
% cd projects/alu_proj
```

one moves to the particular project directory, where one may invoke the Nelsis tool *dblist* by typing

```
% dblist
```

This will list the contents of the project database just as one could type

```
% ls
```

to list the contents of a Unix directory. The output of the *dblist* command will appear on the screen, for example:

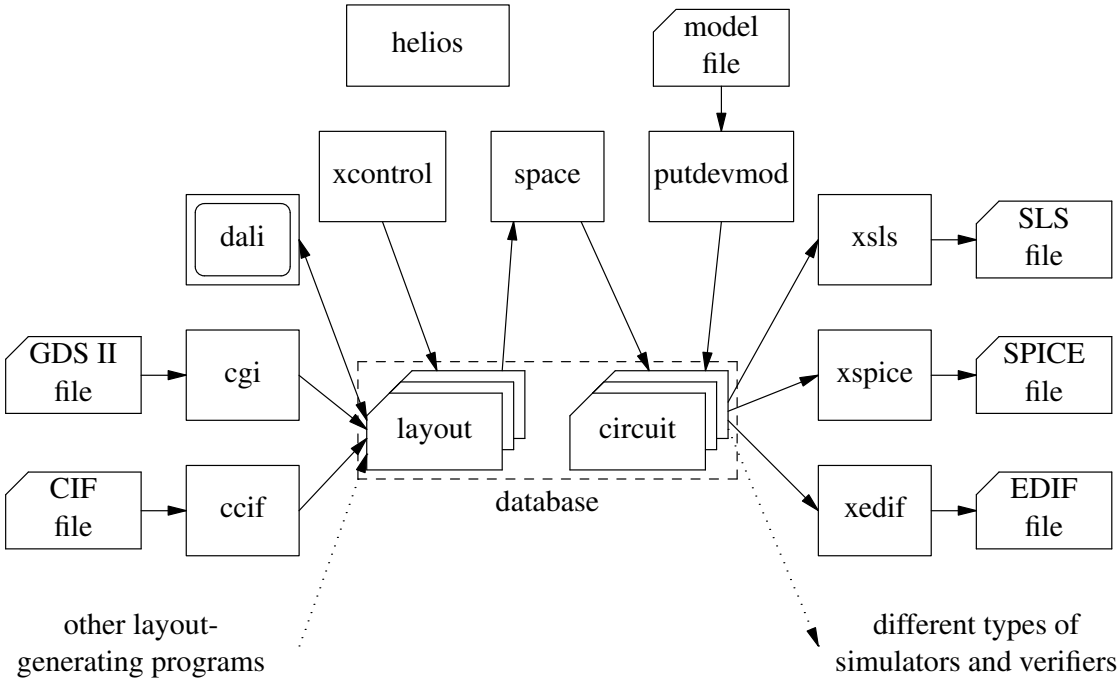
```
% dblist
layout:
feedback        mod2_fb         sel_reg8
latch           rand_cnt        select

circuit:

floorplan:
%
```

The tools are all separate executables. As a result, several tools can be run in parallel and a pipeline of different programs can be set up.

A number of tools have already been mentioned. A flow-diagram for the most important tools is given below. After that one finds a list of all tools that are relevant for extraction, including a short description of each of these tools.



- **mkpr**
  This program is used to create and initialize a project directory. A project directory is the working directory for the extractor and contains subdirectories in which the design data is stored.
- **rmpr**
  This program removes a project directory when it is not needed anymore.
- **addproj**
  This program adds other project directories to the library list of the current project, allowing cells in the library projects to be used in the current project. See also *impcell.*
- **impcell**
  This program makes a cell from a library project available for use in the current project. Only a reference is made, the data is not copied.
- **rmdb**
  This program deletes a cell from a project directory.
- **dblist**
  This program lists the contents of a project directory, i.e. which cells are in it.
- **dbclean**
  This program removes intermediate data, which is generated by some tools and

saved in the project because it can be used more then once, from a project.

- **makeboxl, makegln, makemesh, makesize**
  These programs prepare a layout for extraction, e.g. they expand a layout before flat extraction. For efficiency reasons, their result data are stored into the project. For example, a layout now needs to be expanded only once if it is extracted twice, with different options. These tools are normally not invoked by the designer, but they are automatically invoked by *space.*
- **space**
  This program is the actual layout-to-circuit extractor.
- **tecc**
  This program compiles a user defined technology file for *space* into a format suitable as input for *space.*
- **dali**
  An interactive layout editing program.
- **cgi**
  This program writes a GDS II layout into the project.
- **ccif**
  This program writes a CIF layout into the project.
- **macro**
  This program lists the macro status for layout cells. An instance of a 'macro' cell is always flattened during extraction (not always for all masks). The program can also change the macro status for local cells in old projects. Use the *xcontrol* program for new projects.
- **xcontrol**
  This program is used to set the extraction status for a cell. It supersedes the programs *macro* and *device.* The xcontrol data is stored differently in the project database. This makes it possible to set locally the status of an imported cell.
- **xspice**
  This program retrieves a SPICE netlist from a project.
- **xedif**
  This program retrieves an EDIF netlist from a project.
- **xsls**
  This program retrieves an SLS netlist from a project.
- **xpstar**
  This program retrieves a Pstar netlist from a project.

- **xnle**
  This program retrieves a NLE netlist from a project.
- **xspef**
  This program retrieves a detailed SPEF (IEEE Standard Parasitic Exchange Format) netlist from a project.
- **xspf**
  This program retrieves a detailed SPF (Cadence Standard Parasitic Format) netlist from a project.
- **xvhdl**
  This program retrieves a VHDL (VHSIC Hardware Description Language) netlist from a project.
- **icdman**
  This program is to consult the on-line Nelsis designers manual.
- **putdevmod**
  This program stores information about devices into a project circuit view. It does not set the device status for new projects.
- **device**
  This program lists the devmod status for circuit cells. This devmod status is the device status for old project cells. The program can also change the device status for local cells in old projects. Use the *xcontrol* program for new projects.
- **match**
  This program compares two circuits (e.g. a reference circuit and an extracted circuit) and tells whether or not they are equivalent.
- **highlay**
  This program is used to high-light the layout of one or more conductors and/or devices.
- **helios**
  This program is a Graphical User Interface for the *Space* Extraction System. See the "Space Tutorial Helios Version" for more details.

Apart from the above tools, also other tools are available in the Nelsis System that closely co-operate with *space*, such as different kinds of layout generating programs and simulation programs, such as *simeye* and *sls* (see the manual pages and/or user's manuals of these programs).

2.5 File Structure

In this section we will explain where the different parts of the system are located in the Unix directory hierarchy. In this section (in fact, throughout this document) we assume that these parts are subdirectories of the directory **/usr/cacd**, but this can be different at

your site.

/usr/cacd/share/bin This directory contains the executables. So, if you are going to use *space*, make sure that there is an entry **/usr/cacd/share/bin** in the command search path of your Unix shell.

/usr/cacd/share/man This directory tree contains all on-line manual pages. There is a subdirectory for each of the sections.

/usr/cacd/share/lib This directory tree contains all shared data files such as technology data.

/usr/cacd/share/demo This directory tree contains projects with actual design examples, among which designs presented in this document.

/usr/cacd/share/doc This directory contains the documents (in PostScript form) related to the *Space* System.

2.6 Hierarchy and Terminals

An electrical network or circuit as extracted by *space*, is built up from primitive elements (transistors, capacitors and resistors), subcircuits (when the input layout is hierarchical) and terminals. *Terminals* (sometimes also called *ports* or *pins*) are connectors to the outside world and are used for building up the hierarchical structure of the circuit. The *netlist* specifies the connectivity among all these elements.

For *space* to be able to produce terminals in the extracted circuit, the input layout must also have terminals. Layout terminals are named rectangular areas or points in an interconnect layer of the layout. Space uses the terminals in the layout to create, name and connect the terminals in the circuit according to a one-to-one correspondence between layout terminals and circuit terminals. Terminal names must be unique within a cell.

For flat extraction, only the topmost cell must have terminals. For hierarchical extraction, the topmost cell and all the other cells in the hierarchy must have terminals, unless the cell has a macro status. When a cell has a macro status, the instances of the cell are always flattened, even in case of hierarchical extraction; the macro status is set for a particular cell by using the program *xcontrol*. You can see the macro status also with the program *macro*. *Space* identifies a connection from a cell being extracted (the parent cell) to a subcell (or child cell) when a layout feature of the parent cell touches or overlaps a terminal of the child cell. *Space* identifies a connection between two subcells when a terminal of one subcell touches or overlaps a terminal of the other subcell.

NOTE:
If there are connections between cells that take place not via terminals, use flat extraction only, or set the macro status for all subcells that have this type of connections.

In case of hierarchical extraction without setting the macro status, *space* would not detect such connections. In fact, *space* would be unable to represent those non-hierarchical connections in a hierarchical circuit description. The program will not give a warning in case of such non-hierarchical connections.

Hierarchical extraction is also impossible (in the sense that it results in an incorrect circuit) if the functionality of the child cells is modified by layout features in the parent cells. For example, for hierarchical extraction, when a polysilicon feature in a parent cell overlaps an active area feature in a child cell, this will modify the actual circuit that is defined (by creating a transistor element), while this will have no influence on the circuit that is extracted.

3. Basic Extraction

In this chapter, we will make a tour past most tools and explain the basics of using the *space* layout-to-circuit extractor.

For the examples we will use a hypothetical (but realistic) N-well CMOS process. In the directory /usr/cacd/share/demo there are example designs that we will use.

3.1 Online Manual Pages

The first thing to remember in case one needs some help or information about the tools in the system, is the availability of on-line manual pages of all commands, data-files, etc.. They can be shown on the screen with the command *icdman.* This command works the same as the Unix *man* command, i.e. after showing a screen full of text (24 lines on most ascii terminals) it waits for the user to enter a space and then gives another screen.

For example, information about the command *dblist* can be obtained by typing:

```
% icdman dblist
```

and information about *icdman* itself can be obtained by typing

```
% icdman icdman
```

3.2 Project Creation

To start working with *space*, first a *project directory* has to be created with the command *mkpr*. A project is a Unix directory that contains an IC database. *Mkpr* will generate such a project directory with the name that is given as an argument, after it has asked for the process to be used (e.g. scmos_n, ecpd07 or any other process available in the technology base) and for the value of lambda. It will do all initializations that are necessary at the start of a project. In particular, *mkpr* creates the file and the view directories as was discussed in section 2.1.

Now, make a project with the name "tutorial" and select the *scmos_n* process and enter a λ value of 1 micron. After the project has been created, enter the project directory with the Unix *cd* command.

```
% mkpr -l 1.0 tutorial
available processes:
process id       process name
         1       nmos
         3       scmos_n
         .
         .
select process id (1 - 60): 3
mkpr: -- project created --
% cd tutorial
%
```

When the environment variable ICDPROCESS is set, *mkpr* will not ask for a process name, but it will default attach the new project to the process that is specified by that

variable. ICDPROCESS can specify a process name as well as a path to a process directory. Also, when the process directory contains a file called "default_lambda", *mkpr* will not ask for a lambda value, but it will use the default lambda as specified in that file (therefor we give the option -l).

3.3 Entering a Layout

A layout to be extracted using *space* can be supplied in different formats, like CIF and GDS II. For each layout language, there is a specific tool for entering the layout into the project. For CIF, this is the *ccif* program and for GDS II, this is the *cgi* program. These two programs take a Unix file containing the layout, and store the design data into the layout view of the project.

In the following example we will show how to enter a layout into the tutorial project. In the directory /usr/cacd/share/demo/switchbox there is a GDS II file named switchbox4.gds. Copy this file into the project and invoke the *cgi* command. Assuming your working directory is already the tutorial directory, type

```
% cp /usr/cacd/share/demo/switchbox/switchbox4.gds .
% cgi switchbox4.gds
%
```

Normally, most tools are relatively silent about their operation. They usually only give messages when something goes wrong. You can supply the **-v** option to most of the tools (here: *cgi -v switchbox4.gds*) to obtain a little more feedback on the operation of the tools.

When terminals (pins) are defined as TEXT structures in the GDS file, use the option **-t** with *cgi.*

3.4 Listing the Contents of a Project

Now, you can see what cells are in the project by using the *dblist* command.

```
% dblist
layout:
dec1of4          nan3             switchbox4
dubinv           nan4rout

circuit:

floorplan:
%
```

Clearly, there are 5 layout cells in the project, and no circuit cells. We say that the circuit view is empty.

In order to get more information about the hierarchical structure of the design, you can use the **-h** option of *dblist*.

```
% dblist -h
layout:
1 - switchbox4      (4)
    2 - dec1of4           4 (6)
        3 - dubinv            1 (0)
        3 - nan3              4 (0)
        3 - nan4rout          1 (0)

circuit:

floorplan:
%
```

The first number on a line is the level of the corresponding cell in the cell tree. The second number (if present) is the total number of times (incl. copies) the cell is used, and the third or last number (between parentheses) is the number of instances of sub cells.

3.5 Extracting a Layout

To extract the cell named *nan3*, type

```
% space nan3
%
```

Now, the circuit of *nan3* has been placed in the circuit view:

```
% dblist -c
nan3
%
```

The **-c** option restricts the output of *dblist* to only the circuit view.

The option **-v** can be used to obtain more feedback on what *space* is actually doing. To hierarchically extract the complete tree of *switchbox4* just type the following:

```
% space -v switchbox4
parameter file: $ICDPATH/share/lib/process/scmos_n/space.def.p
technology file: $ICDPATH/share/lib/process/scmos_n/space.def.t
extract hierarchy of switchbox4
preprocessing dubinv
extracting dubinv
nan3 already extracted
preprocessing nan4rout
extracting nan4rout
preprocessing dec1of4
extracting dec1of4
preprocessing switchbox4
extracting switchbox4
space: --- Finished ---
%
```

The circuit now has the same hierarchical structure as the layout:

```
% dblist -h
layout:
1 - switchbox4      (4)
    2 - dec1of4           4 (6)
        3 - dubinv            1 (0)
        3 - nan3              4 (0)
        3 - nan4rout          1 (0)

circuit:
1 - switchbox4      (4)
    2 - dec1of4           4 (6)
        3 - dubinv            1 (4)
        3 - nan3              4 (6)
        3 - nan4rout          1 (8)

floorplan:
%
```

The messages produced by *space* demonstrate the following:

- *Space* reports which technology file it is using. This can be changed with the **-e**, **-E**, **-p**, or the **-P** options.
- *Space* hierarchically extracts the cell named *switchbox4*, its children and their children. For flat extraction, the **-F** option can be used. With hierarchical extraction, only the top cell is extracted when using the option **-T** .
- *Space* employs a preprocessing step before actually extracting a cell. This step involves converting a polygon description into an edge description. The edge files (called gln files) are retained in the database since they can be used in other extraction passes, e.g. with other options.
- *Space* detects when a cell has already been extracted, and skips those cells. (Here, *space* skips *nan3*.) To re-extract a layout for which a circuit already exists, the **-I** option can be used.

By default *space* produces a circuit without any parasitics information. This can be changed by supplying options to the program indicating which type of parasitics must be extracted. The following table lists some common choices, for a more comprehensive description, see the space user's manual or the space manual pages.

-c	extract capacitances to substrate
-C	also extract coupling capacitances, implies -c
-l	also extract lateral coupling caps, implies -C
-r	extract resistances for non-metal interconnect
-z	apply mesh refinement for resistance extraction, implies -r
-k	resistances are only extracted for specified interconnects.
-j	resistances are extracted for all but specified interconnects.

Now, extract switchbox4 again, but use the **-c** option. Also use the **-I** option since the

cells were already extracted. Finally, also use the **-v** option. The usage of this last option demonstrates that *space* does not need to preprocess the cells again.

```
% space -c -I -v switchbox4
parameter file: $ICDPATH/share/lib/process/scmos_n/space.def.p
technology file: $ICDPATH/share/lib/process/scmos_n/space.def.t
extract hierarchy of switchbox4
extracting dubinv
extracting nan3
extracting nan4rout
extracting dec1of4
extracting switchbox4
space: --- Finished ---
%
```

A flat netlist can be produced using the **-F** option:

```
% space -F switchbox4
% dblist -h
layout:
1 - switchbox4      (4)
    2 - dec1of4           4 (6)
        3 - dubinv            1 (0)
        3 - nan3              4 (0)
        3 - nan4rout          1 (0)

circuit:
1 - dec1of4         (37)
    2 - dubinv            1 (10)
    2 - nan3              4 (14)
    2 - nan4rout          1 (18)
1 - switchbox4      (144)

floorplan:
%
```

The *dblist* command reveals that the switchbox4 circuit now has no subcells. Instead, it contains a flat description of the complete circuit. The circuit dec1of4 has become a top-level (root) cell, since it is not instantiated anymore by the switchbox4 circuit.

For flat extraction, *space* needs a flattened gln description, and thus it re-preprocesses the switchbox4 layout. This would have been revealed if the **-v** option was used.

3.6 Retrieving a Circuit

The program *xspice* can be used to retrieve a SPICE netlist from the circuit view of the project. If SPICE transistor models are present in the technology base, they are automatically appended to the SPICE network (the **-o** option can be used to inhibit this). The output of *xspice* can be redirected to a file. Alternatively, the output is stored in a file named *name*.spc, where *name* is the name of the circuit, when the **-f** option is used. (Thus, *xspice -f name* and *xspice name* > *name.spc* are equivalent.)

To obtain a SPICE netlist of nan3, use

```
% xspice nan3 > nan3.spc
%
```

Use the appropriate Unix commands (e.g. *more* or *cat*) to inspect the contents of the file. For simulation, only the input stimuli and *spice* option settings must be added to this file.

3.7 Removing Projects and Data from Projects

When a cell has been extracted, the gln files (or redundant files in general) can be removed with the *dbclean* command. Just typing

```
% dbclean
%
```

will remove all redundant data of all cells in the current project. However, the program also allows less drastic modes of operation, see the manual page of *dbclean*.

All cells in both the layout and the circuit view are removed using

```
% rmdb -a -f
%
```

Without the **-f** option, the program will ask for confirmation in order not to accidentally remove more than actually wanted. Again, less drastic modes of operation are also possible, see the manual page of *rmdb*.

Finally, a project can be removed with the *rmpr* command. Assuming you are in the project directory named tutorial, you can remove that project as follows:

```
% cd ..
% rmpr tutorial
%
```

If the project directory only contains project files and subdirectories (layout and circuit), it is removed entirely. Otherwise it just becomes an ordinary Unix directory.

A project can only be removed if it does not contain any cells. That is, it must be empty like it was just created. Note that a project with the ‘‘force’’ option is always removed.

3.8 Simulation Model Support

The simulation models of the devices that occur in the network are normally specified using the control file of *xspice*. The model parameters may be specified as a function of some layout parameters like the transistor emitter area and perimeter and the transistor base width. The latter is especially useful for the extraction of bipolar circuits (see section 6). Below, we give a simple example that specifies the SPICE model of an nmos transistor.

The default name of the control file for *xspice* is *xspicerc*. First, *xspice* tries to read this file from the current working directory. Otherwise, it tries to open it in the process

directory. The control file may look like:

```
% cat xspicerc
include_library spice3f3.lib

model nenh_0 nenh nmos ()

bulk nmos 0.0
%
```

The above file specifies that the model nenh_0 should be used for the extracted device nenh and that its type is nmos. Note that this only works with known SPICE model types. The bulk voltage of the devices of type nmos is specified to be 0.0 volt. The simulation model itself is included in the library file "spice3f3.lib":

```
% cat spice3f3.lib
model nenh_0 nmos (level=2
                   ld=0.0u       tox=250e-10     nsub=2e16
                   vto=0.7       uo=600          uexp=0.1
                   ucrit=10k     delta=0.2       xj=0.5u
                   vmax=50k      neff=1.0        rsh=0
                   nfs=0e11      js=2u           cj=100u
                   cjsw=300p     mj=0.5          mjsw=0.3
                   pb=0.8        cgdo=300p       cgso=300p)
%
```

When using the program *xspice* to retrieve a circuit description, the following result may be obtained:

```
% xspice mulplex
mulplex

* circuit mulplex 1 2 3 4 5 6
*
*   1 nbulk    2 a        3 b        4 s_a      5 s_b
*   6 out
*
m1 4 2 6 1 nenh_0 w=4u l=1.8u
m2 5 3 6 1 nenh_0 w=4u l=1.8u
* end mulplex

.model nenh_0 nmos (level=2 ld=0 tox=25n nsub=20e15
+ vto=700m uo=600 uexp=100m ucrit=10k
+ delta=200m xj=500n vmax=50k neff=1 rsh=0
+ nfs=0 js=2u cj=100u cjsw=300p mj=500m
+ mjsw=300m pb=800m cgdo=300p cgso=300p)

vnbulk nbulk 0 0.000000V
rnbulk nbulk 0 100meg
%
```

Alternatively, the program *putdevmod* can be used to store a device model description into the circuit database. It uses the ".dev" file (process file for SPICE models) format.

This method does not allow to specify model parameters as a function of layout parameters. However, this method is useful if a device is extracted that can not be recognized from the mask layout combinations and/or if the model for the device consists of more than just a standard transistor model (i.e. if it is described by one or more subcircuits, see section 7).

As an example we consider the following description that is present in a file "nenh.dev":

```
% cat nenh.dev
device nenh
begin spicemod
* terminals  d g s
* bulk       0
* prefix     m
* parameters for n_enhancement NMOS  w/l = 4/4
.model nenh nmos (level=2
+                  ld=0.0u       tox=250e-10  nsub=2e16
+                  vto=0.7       uo=600       uexp=0.1
+                  ucrit=10k     delta=0.2    xj=0.5u
+                  vmax=50k      neff=1.0     rsh=0
+                  nfs=0e11      js=2u        cj=100u
+                  cjsw=300p     mj=0.5       mjsw=0.3
+                  pb=0.8        cgdo=300p    cgso=300p)
end
```

On the first line, this description specifies that a description for the device "nenh" is given. Between the keywords "begin spicemod" on the second line and "end" on the last line of the file, the SPICE information is given. SPICE information that is not meant as direct input for *spice* is given as comment. The description specifies (1) that cell "nenh" is a device that has terminals "d", "g" and "s" (which are used in that order) (these names are also used by *space* to denote the drain, gate and source of a MOS transistor and should therefore be used), (2) that the device requires a bulk voltage of 0 volt (as a result, *xspice* will add a bulk terminal to the device that is connected to a potential of 0 volt), (3) that "m" should be used as a prefix for instance names of the device in a SPICE description (this condition is required by SPICE) and (4) that the device has a SPICE level 2 model description as indicated.

The description is stored in the circuit view of the database using *putdevmod*:

```
% putdevmod nenh.dev
```

After that, a circuit description, in which the simulation model is included, may be retrieved using *xspice*. Note that this only works by a new project if the device status is set with the *xcontrol* program. This can be done as follows:

```
% xcontrol -device nenh
```

4. Extraction of Junction Capacitances

To extract non-linear junction capacitances, these elements have to be specified as junction capacitances in the element definition file of *space.* This is done by including them in a capacitance list where the keyword "junction" is used before the keyword "capacitance" and where a capacitance type is specified after the keyword "capacitance". The following gives an example of the specification of junction capacitances for respectively n diffusion and p diffusion areas.

```
junction capacitances ndif :
    acap_na:  caa       !cpg  csn  !cwn : caa @gnd: 100e-6
    ecap_na: !caa -caa !-cpg -csn !-cwn :-caa @gnd: 300e-12

junction capacitances pdif :
    acap_pa:  caa       !cpg  !csn cwn     : caa cwn: 500e-6
    ecap_pa: !caa -caa !-cpg !-csn cwn -cwn:-caa cwn: 600e-12
```

For junction capacitances, the first terminal mask of the element always specifies the positive node of the extracted element and the second terminal mask always specifies the negative node of the extracted element.

There are several possibilities to represent the junction capacitances in the output circuit, depending on what type of element model is used for the verification (simulation) of the junction capacitance. This is controlled by the parameter *jun_caps* of *space.*

If the parameter *jun_caps* is set to "non-linear" the only difference with the appearance of normal, linear, capacitances in the output circuit is that the extracted capacitance will be of the specified type.

If the parameter *jun_caps* is set to "area" the extracted capacitance will be of the specified type and, moreover, the value of the extracted capacitance will specify the area of the element (if only an area capacitivity is specified for that capacitance type) or the edge length of the element (if only edge capacitivity is specified for that capacitance type). If *jun_caps* is set to "area", and both area capacitivity and edge capacitivity are specified for one junction type, the value of the extracted capacitance will be equal to the extracted capacitance value divided by the area capacitivity (so in this case, effectively, the total vertical and horizontal junction area is extracted).

If the parameter *jun_caps* is set to "area-perimeter" the extracted capacitance will be of the specified type and the area and perimeter of the element will be represented by the instance parameters *area* and *perim* in the database.

If the parameter *jun_caps* is set to "separate" the extracted capacitance will be of the specified type and the area and perimeter of each capacitance that is specified in the capacitance list for that type will separatedly be represented with parameters *area*<*nr*> and *perim*<*nr*> where <*nr*> denotes that it is the <*nr*>-th. area or the <*nr*>-th. perimeter element in the list.

For each junction capacitance type that is extracted, a device model must be defined using either the control file of *xspice* or the program *putdevmod*. Below, we will ilustrate how this is done using a control file for *xspice* since this method offers more flexibility than the method using *putdevmod*.

The control file for the junction capacitances 'ndif' and 'pdif' that are described above may look as follows:

```
include_library spice3f3.lib

model ndif ndif d ()
model pdif pdif d ()
```

where the file spice3f3.lib has the following contents

```
model ndif d (is=2u cjo=100u vj=0.8 m=0.5)
model pdif d (is=10u cjo=500u vj=0.8 m=0.5)
```

The above may be sufficient when only the total junction area need to be specified as an element parameter. The parameter *jun_caps* is then set to "area".

When the horizontal junction area and the vertical junction area need to be specified as separate parameters (e.g. because they have a different grading coefficient), the parameter *jun_caps* should be set to "area-perimeter". Then, the area and the perimeter of the junction region will be extracted as separate parameters. When for the simulation model that will be used these parameters are respectively called "area" and "pj", the control file should then additionally contain:

```
params ndif { area=$area pj=$perim }
params pdif { area=$area pj=$perim }
```

When it is required that for junction capacitance elements the perimeter adjacent to the gate oxide and the perimeter adjacent to the field oxide are specified as separate parameters, the parameter *jun_caps* should be set to "separate", the specification in the element definition file may be as follows:

```
junction capacitances ndif :
    acap_na:  caa  !cpg  csn  !cwn      :  caa @gnd: 100e-6
    ecap_na: !caa !-cpg -csn !-cwn -caa : -caa @gnd: 300e-12
    gcap_na:  cpg !-cpg -csn !-cwn -caa : -caa @gnd: 200e-12
```

and the control file may contain:

```
params ndif { diffarea=$area1 locosedge=$perim1 gateedge=$perim2 }
```

5. Extraction of Drain/Source Area and Perimeter Information

Besides that junction capacitances can be part of the output netlist as diodes, as described in Section 4, they can also be part of the output netlist in the form of drain/source area and perimeter information that is attached to the MOS transistors. Although this method is less accurate to model the distributed RC effects in the drain/source regions, many simulators require this method instead of the previous one.

Area and perimeter information of drain/source regions of transistors is represented by the parameters ad, as, pd, ps, nrs and nrd (see SPICE User's Manual). To extract these parameters, a condition list should be specified in parentheses behind the drain/source mask of the transistor definition, and capacitance extraction should be enabled. The following gives an example of the specification of a drain/source region for a "nenh" transistor and a "penh" transistor.

```
fets :
    nenh : cpg caa  csn : cpg caa (caa !cpg  csn)  # nenh MOS
    penh : cpg caa !csn : cpg caa (caa !cpg !csn)  # penh MOS
```

When an option for capacitance extraction is used with *space,* the SPICE circuit description may contain the following the transistor statements:

```
m1 t_out b_in l_vss nbulk nenh_0 w=6.8u l=1.2u ad=12p as=26.76p
+  pd=11.2u ps=19.7u nrs=0.57872 nrd=0.259516
m2 t_out b_in l_vdd pbulk penh_0 w=12.4u l=1.2u ad=20.5p as=23.36p
+  pd=14.8u ps=10.8u nrs=0.151925 nrd=0.133195
```

The extractor uses heuristics to appropriately subdivide drain/source information over different transistors when all these transistors are connected to the same drain/source region.

6. Bipolar Device Extraction

In this section, we present an example for a control file and a library file for *xspice* (see also section 3.8) for the extraction of bipolar circuits.

Let "npnBW" be the name of a bipolar transistor type that is defined in the element definition file of *space*. Then the following control file may be used for *xspice* to retrieve a SPICE description of a circuit that contains transistors of type "npnBW".

```
% cat xspicerc
# include_library specifies which file(s) contain
# the appropriate model definitions.

include_library  spice3f3.lib

# model indicates which predefined models can be
# used for which group of devices and it includes
# the ranges for area (ae), perimeter (pe) and
# width (wb).

# scaleable model:
#   ae ->  min.  typ.  max.
model bw101a npnBW npn (
        ae 4e-12 8e-12 4e-11
        wb 0.25e-06
        pe (2*$ae / 2.00e-06 + 4.00e-06)
)

# substitution model:
#   ae/pe -> min.  max.
model bw10x npnBW npn (
        ae 4e-12 2e-10
        wb 0.25e-06
        pe 8e-06 6e-05
)
```

Two types of models are used for the "npnBW" transistor: "bw101a" and "bw10x". Which model is chosen for a particular transistor in the circuit depends on the actual value of some transistor parameters that are extracted for it: its emitter area (ae), its emitter perimeter (pe) and its base width (wb). For both transistor models, the base width must be equal to 0.25e-06. Model "bw101a" is selected if ae is between 4e-12 and 4e-11 and pe is equal to 2*$ae / 2.00e-06 + 4.00e-06. Model "bw10x" is selected if ae is between 4e-12 and 2e-10 and pe is between 8e-06 and 6e-05.

Note that the above bipolar model choices only works for some known model types and known extracted parameter names.

The SPICE models are specified in the file "spice3f3.lib":

```
% cat spice3f3.lib
unity Q_electron    1.602e-19
unity N_intrinsic   1.045e+20
unity Gummel_base   7.500e+06
unity C0s_wn_bw     1.900e-03
unity C0e_wn_bw     2.800e-09
unity C0s_bw_epi    0.290e-03
unity C0s_bn_sub    0.151e-03

model bw101a npn (Is=0.018f Bf=117 Nf=1 Vaf=55 Ikf=4.1m Br=4
                  Nr=1 Var=4 Ikr=45u Rb=600 Irb=0.15m Rbm=30 Re=14
                  Rc=200 Xtb=1.5 Eg=1.17 Xti=2.5 Cje=50f Vje=0.78
                  Mje=0.28 Tf=20p Cjc=75f Vjc=0.67 Mjc=0.32 Xcjc=1
                  Tr=100p Cjs=0.24p Vjs=0.45 Mjs=0.26)

model bw10x  npn (Is=($Q_electron*$N_intrinsic/$Gummel_base)*$ae
                  Nf=1 Ikf=3.00e+07*$ae+6.00e+01*$pe Bf=117 Br=4
                  Vaf=55 Var=4 Ikr=5.00e+07*$ae+1.00e+02*$pe Xtb=1.5
                  Eg=1.17 Xti=2.5 Cje=$C0s_wn_bw*$ae+$C0e_wn_bw*$pe
                  Vje=0.78 Mje=0.28 Tf=20p
                  Xtf=(4.70e-02*$ae+1.90e-02*$pe)^2 Tr=100p Mjc=0.32
                  Vjc=0.67 Cjc=$C0s_bw_epi*$ae Cjs=$C0s_bn_sub*$ae
                  Vjs=0.45 Mjs=0.26)
```

For model "bw101a", the model parameters have a fixed value and are chosen optimal for ae = 8e-12, wb = 0.25e-06 and pe = (2 * 8e-12 / 2.00e-06 + 4.00e-06) (see the specification in the control file). For transistors that have different parameter values - but that are within the range specification that is given in the control file - model "bw101a" will be used in combination with a scale factor.

For model "bw10x", the model parameters are a function of the transistor parameters ae and pe. In this case, *xspice* will generate a different model for each different set of parameter values that occur in the circuit.

7. Special Device Extraction

By using *putdevmod* (see section 3.8) it is possible to extract special devices that can not directly be recognized from the mask combinations in the layout. The basic idea is to define a layout cell for each type of special device, including its terminals, and then store a corresponding model description in the circuit view of the database by using *putdevmod*.

Suppose we have a layout cell of a special device "lpnp" that has 6 collector, 2 base and 2 emitter connections. The terminals are called c1, c2, c3, c4, c5, c6, b1, b2, e1 and e1. Then, a corresponding device model description may be created for this device as shown below.

```
% cat lpnp.dev
device lpnp
begin spicemod
* terminals c1 c2 c3 c4 c5 c6 b1 b2 e1 e2
* bulk 0
* prefix x

.subckt lpnp 1 2 3 4 5 6 7 8 9 10 11
rn1 7 8 20
c7 7 0 0.001f
c8 8 0 0.001f
q1a 1 8 9 b_spnp 0.5
q1b 2 8 9 b_spnp 0.5
q1c 3 8 9 b_spnp
q1d 4 8 10 b_spnp 0.5
q1e 5 8 10 b_spnp 0.5
q1f 6 8 10 b_spnp
c1 1 0 0.001f
c2 2 0 0.001f
c3 3 0 0.001f
c4 4 0 0.001f
c5 5 0 0.001f
c6 6 0 0.001f
dq1 11 8 c_diojcs1a 2
.ends

.model b_spnp pnp(is=1.2e-15,bf=60,vaf=100,ikf=3e-4,
+ise=24e-12,ne=3.6,br=6,var=18,ikr=1e-4,isc=1e-15,
+nc=1.15,rb=100,rbm=50,rc=80,re=10,cje=0.2e-12,
+mje=0.33,vje=0.55,cjc=0.96e-12,mjc=0.33,vjc=0.55,
+tf=4e-8,xti=3.3,eg=1.16,xcjc=0.8)

.model c_diojcs1a d(is=1.26e-14 rs=200 cjo=3.1e-12 vj=0.50 m=0.35)

end
%
```

This description specifies that cell "lpnp" is a device that has the same terminals as the layout cell and that it should be included in a SPICE network description with an instance

name that starts with the prefix "x". A bulk potential of 0V is specified, which is the appropriate potential for the (extra) terminal 11 that is also specified for the device model. Program *xspice* will correctly connect this bulk terminal when the device is retrieved from the database. The interior of the device model is described by 6 transistors, 1 diode, 1 resistor and 8 capacitors.

After storing the above description in the circuit view by using the program *putdevmod*, the description can be used to extract circuits that contain transistors of type "lpnp".

```
% putdevmod lpnp.dev
%
```

During extraction, the extractor will then recognize that cell "lpnp" is a device, that it is not necessary to extract the contents of this device, and that it should be included in the network description as a network primitive. For a new project, you must also define the device status with the *xcontrol* program. Note that by new projects the devmod data is only used by the program *xspice* after that the device status is set.

When a cell that is a device is also defined as a macro, the cell will be included in the extracted network as a primitive (just as in the normal case), but the layout of the cell will additionally be flattened in a father cell prior the extraction. This allows the layout of the device to be used as feed-throughs for other connections and to make connections to the device not only at the terminal positions.

8. Layout versus Schematic

The program *match* is used to compare an extracted circuit (the actual_network) against a reference circuit (the nominal_network). As a reference circuit, take the SPICE description swbox_ref.spc that is present in the directory /usr/cacd/share/demo/switchbox and add it to the ‘‘tutorial’’ database using the program *cspice*:

```
% cp /usr/cacd/share/demo/switchbox/swbox_ref.spc .
% cspice swbox_ref.spc
%
```

As an actual circuit, take the flat layout description switchbox4_f (the hierarchical description switchbox4 could also be used but it is easier to modify the flat layout description), put it into the database and extract it.

```
% cp /usr/cacd/share/demo/switchbox/switchbox4_f.gds .
% cgi switchbox4_f.gds
% space switchbox4_f
%
```

Then, use the program *match* to compare the extracted circuit against the reference circuit:

```
% match swbox_ref switchbox4_f

match: Succeeded.

%
```

The result of *match* shows that the reference circuit and the extracted circuit are identical.

Next, make an error in the layout of switchbox4_f (e.g. remove some metal) with a layout editor and run the programs *space* and *match* again.

```
% space switchbox4_f
% match swbox_ref switchbox4_f

match: Failed.

%
```

This result indeed shows that the circuits are not identical. The network parts that have been matched and the network parts that have not been matched can be inspected by using the option **-fullbindings** with *match*.

9. Layout Back-Annotation

Layout back-annotation is supported in three different ways. First, the names of the terminals (and the instances) in the extracted circuit are equal to the names of the corresponding terminals (instances) in the layout. Second, the option **-t** can be used to add positions of devices and sub-cells to the extracted circuit. Third, the program *highlay* may be used to high-light the layout of one or more conductors and/or devices. The latter program may also be used in combination with the circuit comparison program *match* to for example high-light all matched or unmatched conductors.

To demonstrate the use of *highlay* in combination with the circuit comparison program *match*, consider the example from the previous section. Assume that the reference circuit swbox_ref has been added to the database and assume that an error is present in the layout of switchbox4_f. Now, run the program *space* with the option **-x** and run the program *match* with the options **-edif** and **-fullbindings** . The last two options cause *match* to write a binding table into the database that can be used as input for *highlay.*

```
% space -x switchbox4_f
% match -edif -fullbindings swbox_ref switchbox4_f

match: Failed.

%
```

Then, the nets that have not been matched (the deficient nets) are selected for high-lighting by running the program *highlay* as follows:

```
% highlay -d -n -v switchbox4_f
Selected nets:
38
%
```

Next, use a layout editor to read in the cell HIGH_OUT that has been generated by *highlay* and inspect the unmatched conductors.

```
% dali HIGH_OUT
```

10. Sea-of-Gates Circuit Extraction

For semi-custom circuits like sea-of-gates circuits, often an image is defined that contains a regular pattern of MOS transistors and/or bipolar transistors. On top of this image, a metal pattern is placed that connects all relevant devices. A design of such a circuit may for example be represented by the following hierarchical tree.

```
% dblist -l -h
1 - reg4            (8)
    2 - latch             4 (2)
        3 - image_c           2 (0)
    2 - select            4 (1)
        3 - image_c           1 (0)
%
```

The image cell "image_c" is the smallest tile in the image that can be repeated to give the complete image. In order to extract accurate capacitances, the image cell should be flattened in the cells "latch" and "select", that only contain metal wires. For flat extractions, the layout of the image cell will always automatically be flattened. For hierarchical extraction, the flattening of the image cell can be achieved by setting the macro status for the cell "image_c".

The macro status is set for a particular cell by using the program *xcontrol.*

```
% xcontrol -macro image_c
%
```

Then, when the circuit is extracted in hierarchical mode, a circuit is produced in which "image_c" has been flattened but in which the rest of the hierarchy has been preserved.

```
% space -c reg4
% dblist -c -h
1 - reg4            (8)
    2 - latch           4 (18)
    2 - select          4 (8)
%
```

This, however, only works when all connections to and through the sub-cells latch and select are made via the terminals of these cells. When this is not true, flat extraction has to be used.

The program *ghoti* can be used to remove unused (i.e. unconnected or partly connected) components from the extracted network.

If a gate-level circuit description has to be obtained from the sea-of-gates layout, all cells that represent gates should be set to the library status (celltype). This is done as follows:

```
% xcontrol -library latch select
%
```

Note: For old projects you must set the device status using the *device* program.

Now, no matter whether a hierachical of flat extraction is performed, the cells "latch" and "select" will always be primitives in the extracted network. If the layout that is inside these cells also needs to be flattened - because of connections to or through the cells not only via the terminals - also set the macro status for these cells. This can be done by setting the interfacetype free (or freemasks) as follows:

```
% xcontrol -free latch select
%
```

Note: For old projects you cannot set an interfacetype, but must set the macro status using the *macro* program.

CONTENTS