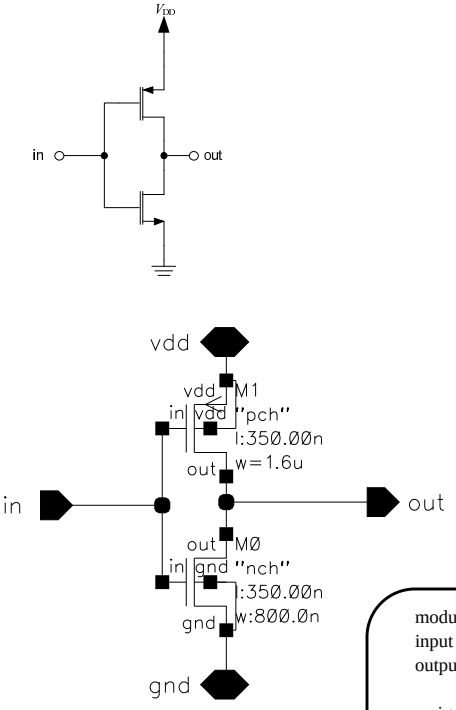
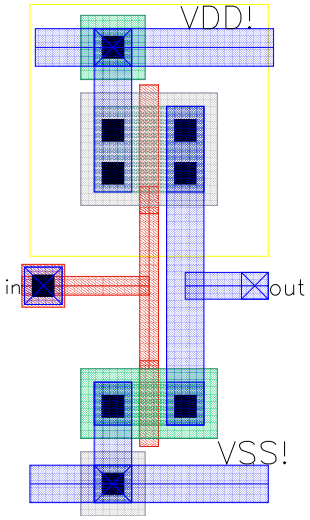
VLSI User's Manual

2008 Edition






```
module inverter (in, out);
input in;
output out;

assign out = !in;

endmodule; //module inverter;
```

University of Toronto

VLSI User's Manual

2008 Edition

VLSI Research Group
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada

Preface

This manual was prepared for use in the laboratory section of the course "VLSI Design Methodology," taught in the Department of Electrical and Computer Engineering at the University of Toronto. The laboratory section of this course introduces students to commercially available VLSI design software from Cadence and Synopsys, with a special emphasis on the TSMC CMOSP35 (0.35-micron CMOS) and TSMC CMOSP18 (0.18-micron CMOS) technology available through the Canadian Microelectronics Corporation (CMC, http://www.cmc.ca).

Some of the material included here was modified from Cadence on-line documentation, Synopsys training course materials, CMC on-line documentation, and from deliverables for a joint CMC/Micronet BiCMOS contract. Although every effort has been made to ensure the correctness of this material, updates to design tools and technology design-kits necessitate occasional updates to the content of this document. Corrections and updates will be made available at

`http://www.vrg.utoronto.ca/vrg/DesignManual2008`

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 Introduction: VLSI Design Methodology

This manual is intended to serve as a concise reference guide to VLSI design at the University of Toronto. We assume that the reader has had some background in electronics and digital circuits design.

A variety of design styles or alternatives is available to integrated circuit designers. Some examples include

- the gate array approach, where the designer specifies the interconnections to be completed on a partially-fabricated wafer;
- the field-programmable gate-array approach, where a fully-fabricated packaged chip is electrically programmed to perform a specific function;
- the "full custom" approach, involving the full customization of all mask layers.

The tools and techniques described within this manual will assist the designer in the last of these topics, known as hierarchical design methodology.

The primary motivation for hierarchical design is the fact that thus far, VLSI processing technology has advanced faster than design capability. VLSI technology permits the realization of very complex circuits and there is, therefore, a need for design methods that can manage the complexity in design and thus fully exploit available VLSI capabilities.

Hierarchical design involves the successive decomposition of a complex design into a series of lower complexity problems which can be solved independently. It is also known by such terms as structured design, top-down design and the Mead-Conway approach. Besides the abstraction design philosophy, the principle of regularity (the use of a few parts repeated frequently) is also applied as opposed to customizing each piece of design. In this case, what might be lost in circuit performance is more than compensated for in the reduced circuit development time and, because most of the parts have been predefined and used before, there is less chance for error in the circuit. The advantages of structured and regular design include easier human interpretation and easier design verification.

1.2 This Manual

This chapter describes some of the commercial software tools available here at the University of Toronto. Subsequent sections introduce some of the tools used for schematic entry, physical layout, verification, and simulation. Next, we describe some of the Cadence and Synopsys tools used for design using higher levels of abstraction, allowing designers to perform behavioural or structural simulations of large systems, as well as synthesis and optimization. The last chapter describes the use of Cadence *First Encounter* to perform automated place-and-route of a synthesized design.

1.3 The CMC Sustainable Technology Configuration

Most of the computer-aided design tools and technology-specific design kits accessible to designers have been made available by *CMC Microsystems* (formerly the Canadian Microelectronics Corporation, or **CMC**), and are installed on the `/CMC` disk partition which should be accessible from all Solaris and Linux computers on the EECG and VRG computer networks within the Department of Electrical and Computer

Engineering. This disk pack is also known as the **STC**, or *Sustainable Technology Configuration*, a methodology for keeping design software at Canadian universities participating in microelectronics research up to a standard level.

The CAD tools are installed under the directory `/CMC/tools` and process technology design kits are installed under the directory `/CMC/kits`. Each tool and design kit is installed in a suitably-named directory with an extension corresponding to its version number; a symbolic link is created, named after the tool or technology (with no version number), pointing at the currently supported version. Symbolic links in appropriate tool library directories point to specific technology design kit directories. The tools are accessible to all researchers, but technology design kits are normally protected such that only designers who have signed a *Confidential Information and Intellectual Property Agreement* for the specific technology, as discussed in section 2.4 "Obtaining Access to Technology Information" on page 2-4.

1.4 Versions of Tools

New versions of CAD tools are frequently updated by the vendors, to address problems and to incorporate new features; new releases are typically available on a quarterly basis. Examples in this manual were tested with the following releases of software tools:

- **Cadence IC.5141.USR4**: IC tools, for schematic, layout, verification, analog environment
- **Cadence IUS.583.USR7**: NClaunch/NCverilog/NCvhdl/NCsim for Verilog/VHDL simulation
- **Cadence SOC.52.USR5**: SOC Encounter, for place-and-route
- **Synopsys vcs-mx.X-2005.06**: for HDL simulation (optional)
- **Synopsys syn_vX-2005.09**: for design synthesis
- **Mentor calibre_2007.3_27.17**: for design verification
- **Mentor dft_2007.3**: for Design For Test tools (optional)

1.5 Versions of Design Kits

Technology design kits are updated periodically, but far less frequently than the CAD tools used for designing microchips. The examples in this manual were tested with these technology releases:

- **cmosp35.4.3**: TSMC 0.35-micron CMOS technology
- **cmosp18.5.2**: TSMC 0.18-micron CMOS technology
- **artisan.3.0**: Standard-cell and I/O pad libraries compatible with **cmosp18**

Chapter 2 Cadence Tools: A Beginner's Guide

The objective of this chapter and the next three chapters is to help the designer become familiar with the Cadence family of CAD tools. This chapter covers the bare basics: how to set-up your account to access the tools and technology libraries; how to set-up your account so that your environment variables point to directories and libraries used by Cadence tools; and, most importantly how to launch and exit the Cadence integrated IC design environment.

The Cadence design environment supports all of your design activities beginning with schematic capture, and all the way through to physical design layout, verification, and hopefully design tape-out.

The integrated circuit designer typically begins by drawing a transistor schematic for their design. Such a schematic may be used to generate a netlist, which is just a file listing the devices used in a design and a list of the connections existing between device pins. This netlist is used to run a circuit simulation in order to verify correct function and operation.

This schematic can be used to guide physical mask layout, and as a guide for inter-device and inter-block connectivity. The layout versus schematic (LVS) comparison tool can be used to run automated checks on the schematic and corresponding layout to check whether the two representations of the same circuit agree. Software can generate a netlist for the layout by first detecting devices using technology-specific rules to recognize devices based on how various mask layers overlap, and then detecting the inter-connectivity between devices by annealing pieces of metal layers, vias, etc. into nets.

Design rule checking (DRC) may be run on the layout view to ensure that there are no violations of process rules, which are defined by the process engineers at the fabrication facility (known familiarly as the "foundry") to guarantee that a sufficient percentage of ICs are functional after manufacturing. Design rules restrict transistor sizes (e.g. minimum channel length, minimum diffusion area), minimum spacing between geometries (e.g. wire-to-wire distance, contact to gate distance), and other geometric dimensions that can impact device operation or performance.

The transistors and other devices used to create the schematic might have slightly different characteristics from the devices used in the schematic. This can be caused by a number of factors, but is usually due to the fact that the transistors used in the schematic use only an estimated diffusion area size. Moreover, the wires in our "ideal" schematic are just that: wires; they have no associated capacitance or resistance. A completed layout can be used to generate a more indicative netlist for your completed design. This netlist will include all of the parasitics that are determined by analyzing the circuit's layout. These parasitics are extracted using technology-specific rules to recognize devices (e.g. NMOS and PMOS transistors) based on the overlap of various mask layers. A parasitic extraction uses technology-specific rules which are used to compute the capacitance seen by each wire trace and the resistance of this wire trace. It is important to note that the capacitances and resistances extracted from a layout might not have been intended by the designer; rather, they are caused by the intrinsic properties of wires, parallel wires and wires running over a ground plane (e.g. the substrate). Post-layout simulation, a simulation of the parasitic-extracted design, is usually required before a final sign-off, to guarantee proper circuit operation before it is actually manufactured.

Electrical Rules Checking (ERC) may be performed on the extracted layout view of a design, to ensure that standard rules have not been broken, e.g. power and ground wires are not shorted together, no floating nodes exist.

The final steps of the design process requires the conversion of the layout view of the design into a format acceptable in commercial fabrication houses, "GDSII" (or "GDS" or "Stream") format. For historical reasons, the conversion to this format is known as "the tape-out" because integrated circuit designs were typically transferred from design computers to the fabrication facility via magnetic tape before network connectivity became ubiquitous.

2.1 Setting-up Your Environment for CMC CAD Tools

The majority of the CAD tools described in this manual may be set up by using some built-in `csh(1)` or `tcsh(1)` commands; most tools have associated "source" files that may be used to set up your design environment for those tools. Because other shells like `bash(1)` and `sh(1)` use incompatible syntax, we recommend that you not use these for your default login shell. [See on-line manuals for the `chsh(1)` command to change your default shell.]

You can ensure that Cadence tools are included in your account set-up by issuing the command

```
% echo $path
```

The system should respond by displaying a list of paths that are searched when you type a command at the system prompt. If your system is set-up to access Cadence CAD tools, and other necessary CAD tools, then you should see these (or similar) directory paths displayed, among others:

```
/CMC/tools/cadence/IC/tools/bin
/CMC/tools/cadence/IC/tools/dfII/bin
/CMC/tools/cadence/IC/tools/dfII/local/bin
/CMC/tools/meta/Latest/bin
/CMC/tools/synopsys/sim/sparcOS5/sim/bin
```

(You might see version numbers following /`CMC/tools/cadence/IC...` indicating that a specific version of the tools are being used; this is okay.)

If these paths are not displayed, then your account has not been configured yet. To remedy the situation add the following lines to the `.cshrc` file in your home directory:

```
if ( -e /CMC/tools ) then
  source /CMC/tools/CSHRCs/Cadence
  source /CMC/tools/CSHRCs/Cadence.IUS
  source /CMC/tools/CSHRCs/Cadence.SOC
  source /CMC/tools/CSHRCs/Synopsys
endif
```

> **Note 1**: The first line tests for the accessibility of the network-accessed tools. The "source" commands will only succeed if there are no network problems and the server is accessible.
>
> **Note 2**: Most CAD tools described in this manual are available on Solaris and Linux platforms, and are accessible by using a single "source" file that does appropriate set-up depending on the platform you are using.

2.2 On-line Documentation

Full product documentation for the Cadence family of tools, including fully searchable manuals, are available on-line using the command

```
% cdsdoc
```

Issuing this command will open a window showing a list of folders, each folder containing documentation about a Cadence product. Double-click on the folder to open it and examine the available documentation. Double-click on any document and it will be opened using Netscape or your default browser; a new browser process or an existing browser process may be used.

Local information on Cadence products can also be accessed from the Cadence IC tool command interpreter window (CIW) by choosing VRG Info. News, from Cadence technical contacts and local staff will also be accessible using this menu link. Technology-specific documentation, named for the technology being used, is often provided via another drop-down menu in the Cadence IC tool CIW. Any changes made to the `/CMC` disk partition are recorded in the file `/CMC/CHANGES.vrg`. Items are listed in reverse-chronological (recent changes first) order.

In addition, most of the process technology design kits, stored under `/CMC/kits`, will also have at least one file with the prefix "`==README`" describing any local modifications that were made. Some of the more complex technologies include files with names `==README.*.quickstart` -- be sure to examine these for hints and for answers to some frequently asked questions related to these technologies.

2.3 Customizing Your Cadence IC Tool Design Environment

2.3.1 Customizing your CadenceIC tool start-up environment: The `.cdsinit` file

New VRG accounts are pre-configured with a simple initialization file `.cdsinit,` which can be found in your home directory. This default configuration file will also load a local configuration file `.cdsinit.local` which it searches for in the directory where you start your Cadence session. This set-up allows you flexibility in the customization of your design environment for each technology you might work with. Users with EECG accounts should copy this default configuration file to their home directory using

```
% cp  /CMC/kits/VRGlocal/cdsinit  $HOME/.cdsinit
```

2.3.2 Customizing Cadence IC tool environment variables: The `.cdsenv` file

Command options and values for environment variables used by some of the Cadence tools can be set up in the `.cdsenv` file. Cadence tools will first read the default version of this file, located at

```
/CMC/tools/cadence/IC/tools/dfII/local/.cdsenv
```

and then it will read additional information from the `.cdsenv` file located in your home directory, so you can override any of the built-in settings. Modifying the `.cdsenv` file in your home directory might be useful after you obtain some experience with the Cadence tools. For example, to select a different default simulator or waveform viewer, you may use the following settings.

```
asimenv.startup  simulator      'string  "spectre"
asimenv.startup  cds_ade_wftool 'string  "awd"
asimenv.startup  projectDir     'string  "./simulation"
```

(This is an example only, and is used to illustrate the contents of the .cdsenv file. Do not make these changes at this time.)

2.4 Obtaining Access to Technology Information

Process technology design-kits are available for the following technologies: bicmos, cmosp35, cmosp18, cmrf8sf, cmos90nm, cmos65nm, and a few other specialized technologies. Some of these technologies (bicmos) are relatively old and have no special access requirements. Access to all other technologies is severely restricted, and may only be set up after designers sign a "*Confidential Information and Intellectual Property Agreement*" for each. For details, see

```
http://www.vrg.utoronto.ca/UofT_Access_Only/TechnologyAccess.html
```

NOTE: Non-disclosure agreements are legal documents. Your signature indicates that you will comply with the terms of the agreement. Any information to which you are given access after signing a non-disclosure agreement must remain confidential and can not be copied from VRG or EECG computers, at risk of expulsion from the University and/or criminal prosecution.

In this manual, we will be using the cmosp35 and cmosp18 technologies for design examples. You will need to sign a Confidential Information and Intellectual Property Agreement with The University. Although the techniques described in this manual are generally applicable to other technologies, the emphasis here will be on CMOSP35 and CMOSP18. Design-kits for other technologies might not support all of the features described here, or may require modifications to these techniques.

2.5 Creating a Project Directory

Each technology requires some technology-specific libraries, hence it is strongly recommended that you create a separate subdirectory under your UNIX login account for each technology you use, or for each major project, if you wish. You should start the Cadence IC tools from within a technology-specific subdirectory to ensure proper access to technology libraries. For example, to initialize a work-directory for CMOSP35 designs, I might type (in a UNIX window):

```
% cd $HOME             # change to my HOME directory
% mkdir CMOSP35        # create a new work directory named "CMOSP35"
% chmod 700 CMOSP35    # protect this directory!
% cd CMOSP35           # change directories
% startCds -t cmosp35 # start Cadence tool with "cmosp35" kit
```

In subsequent CMOSP35 design sessions, I only need to type

```
% cd $HOME/CMOSP35    # change directories
% startCds -t cmosp35 # start Cadence tool with "cmosp35" kit
```

The command *startCds* is a shell script which creates a file `cds.lib` only if it does not already exist, in the directory from which you invoked the script. This `cds.lib` file holds a mapping between the names of design libraries and their actual locations in the UNIX filesystem hierarchy. You must have

signed any non-disclosure forms (or other forms required for access to a technology) and have been granted access to the technology information via UNIX file and group permissions before attempting to use the *startCds* script successfully with a restricted technology.

2.6 Starting a Design Session

Assuming that you now have a correct account setup and access to the technology has been granted, you can start a Cadence design session. Change directories to your technology-specific or design-specific work directory and start a session using the command:

```
% startCds -t <technology>
```

The `-t <technology>` option is used to specify the technology you wish to use. For CMOSP35, use `cmosp35` as the technology option and for CMOSP18, use `cmosp18`. Upon successful start-up of the software, a Command Interpreter Window (CIW) will be displayed. This is your Cadence session console, and will be used to display any warning or error messages from the software. (A record of the session will also be stored in a file in the subdirectory $HOME/CDSlogs; this may be useful for VRG staff to help trouble-shoot any problems you encounter.)

Near the top of the CIW is a set of titles corresponding to pull-down menus (e.g. File, Tools, Options, etc.). Most technology design-kits include a menu item specific to the technology or to the company who provides the technology. As well, the "`VRG Info`" menu shows some potentially useful information. At the far right edge of the menu banner in the CIW (and on all other windows in most Cadence tools) you will see a "`Help`" menu with context-sensitive information.

Earlier, we mentioned that local customization of Cadence tool start-up scripts is recorded in the text file

```
/CMC/CHANGES.vrg
```

If the contents of this file was updated since your most recent design session began, a pop-up window displaying its contents will appear on your screen when you start a new session. (Needless to say, this window will always appear the first time you start a Cadence session.)

2.6.1 Potential problem: "*Warning: ridiculously long PATH truncated*"

As mentioned earlier, you might need to manually add the following line to the `.cshrc` file in your home directory:

```
source /CMC/tools/CSHRCs/Cadence
```

If the addition of this line introduces errors upon login such as "*Warning: ridiculously long PATH truncated*", one work-around is to edit the `.cshrc` file again and comment out three specific lines. Near the beginning of the file there are two lines testing and setting a shell environment variable named `LEVEL`. Change these two lines by adding a ('#') character to the beginning, that is, change

```
if ( ! $?LEVEL ) then
   setenv LEVEL 1
```

into

```
# if ( ! $?LEVEL ) then
#   setenv LEVEL 1
```

Further down in the file, you will have to comment-out the corresponding `else ... endif` lines and whatever is between them. If you have a recent `.cshrc` file, you will change

```
else
  # count the depth ...
  set level=$LEVEL ...
endif
```

into

```
#else
#  # count the depth ...
#  set level=$LEVEL ...
#endif
```

2.6.2 Displaying a design session on a Windows PC

Cadence design sessions may be displayed on a Windows PC only if the PC is running an X-Windows emulator (e.g. Hummingbird or Cygwin) or VNC. Older versions of Cadence tools required that your display adapter settings were set for *8-bit (pseudo-color)* or to *24-bit (true-color)*, but these limitations have been removed in modern releases of the tools.

2.7 Design Kit Layout

The Cadence design tools available here do not include any technology data. The actual technology data is obtained from various manufacturing facilities, and are set up in the form of a technology "design kit", or "process design kit" (PDK) saved under the `/CMC/kits/` directory.

2.7.1 Older Design Kits

Originally, most design-kits used at University of Toronto were obtained from technology vendors with the assistance of CMC Microsystems, and were set-up following a template under the directory `/CMC/kits/<technology>.<version>`, where "`<technology>`" is the name of one of the CMC-distributed design kits, or it may be the name of a technology accessible only to specific researchers under full non-disclosure agreements with commercial fabrication facilities. As new versions of the design kits are released, new directories are created with different "`<version>`" numbers. A link is created to the currently-supported version directory and is named without the "`.<version>`" suffix.

A typical layout of a technology design-kit will include the following files and directories:

- README — A file describing the technology and its features. Information here will be useful to designers and to system administrators.
- cds.lib — A file mapping specific technology-related library names to the UNIX file-system hierarchy. This file is copied to your work directory upon your first invocation of Cadence tools for a specific technology, only if a file by that name does not already exist, and only if you have been granted access permission to the technology. (If you have not been granted access to the technology before starting Cadence, you will see a warning message in a pop-up window indicating ``Problem reading the cds.lib file. Maybe there is a problem with the cdsd daemon..." In this case, exit from Cadence, remove the cds.lib file from the work directory if it is empty, and reread Section 2.4 "Obtaining Access to Technology Information" on page 2-4.

- display.drf A file loaded into your Layer Selection Window (LSW) during Cadence sessions, indicating colors and stipple-patterns used for displaying mask layers
- doc/ Documentation directory
- dfII_lib/ Directory holding Cadence dfII technology libraries, for device schematic symbols, standard cells, and sample structures
- models/ Directory containing simulation models for HSPICE, SPECTRE, and Verilog simulations
- skill/ Directory containing customized SKILL programs required for this technology
- synopsys Directory containing files related to Synopsys tools
- <technology>.tf A technology file which is normally used to compile the libraries stored under the dfII_lib/ directory
- <technology>.strmMapTable A file used during conversion of layout data for "tape-out"

2.7.2 Newer Design Kits

In recent times, we have been obtaining more design kits without the assistance of CMC Microsystems. These kits are often configured according to the specifications of the CAD groups within the companies providing the technologies, and our agreements with these companies often prevent any modifications within the design kit directories. However, we still try to provide a set-up similar to that of the older kits.

2.8 Cadence Design Storage Structure

2.8.1 Design storage in Cadence

The hierarchy of data storage is organized as follows:

- A **library** contains a number of related designs called cells.
- A **cell** will have different representations called views.
- A **cellview** could be one of layout, schematic, extracted, symbol etc.

For example, in a design library *lib_tutorial*, you may have cells *circuit1* and *circuit2*. The cell *circuit1* may have several representations, including a *layout* cellview and a *schematic* cellview. The cell *circuit2* may contain a *symbol* cellview and a *schematic* cellview.

NOTE: This hierarchical structure is managed by Cadence. Although the actual storage structure in your UNIX account corresponds to this hierarchy, **it is strongly recommended that any library management (renaming cells, deleting cells or libraries) be done only from within the Cadence system**, using the Library Manager, to avoid confusing Cadence.

NOTE: Many of the Cadence tools and technology-related SKILL routines rely on specific names for cellviews, so changing these from their defaults is not recommended; use only the word **layout** for mask-layout cell views; use only **schematic** for schematic cell views, and so on.

2.8.2 Data management

All existing libraries in the library path will be displayed in a ***Library Manager*** window, which can be invoked by typing the **F6** function key while the mouse cursor is in the CIW. This is where you access all your designs. The ***Library Manager*** displays all libraries and files, allows you to open, copy and delete files. It is possible to copy entire designs from one library to another. The ***Library Manager*** can also be

invoked by selecting **Tools→Library Manager** in the CIW menu. There is also a ***Library Browser*** that can be invoked by selecting the Browse option in the *Open File* form. This tool does not allow any data manipulation but you can use it to find and open a cell view.

2.8.3 Library path

This is a list of UNIX directories where Cadence looks for the design libraries previously created. The pre-defined search path is set automatically as required by the technology initialization routines; the current working directory (where you start your Cadence session) is included in the default search path.

Note: The libraries displayed in the Library Browser and Library Manager can be located in many different UNIX directories.

Modifications to the library path may be made using the **Tools→Library Path Editor**. This provides a mechanism by which you may modify the paths to existing libraries, or add new libraries and paths as they become available.

2.8.4 Compacting Library Data

Defragmenting, or compacting, design libraries allows you to remove the free space which can accumulate in design files, and thus reclaim large amounts of disk space. To invoke the defragmentation program, execute the following UNIX command:

```
squishDB [-v] [-Log] [-Backup] [-Path searchPath] [-Lib libraryName]
```

where

- **-v** sets the message level to verbose
- **-Log** logs all messages into a file named ./translate.log
- **-Backup** keeps a backup of each file compacted. Default is to keep no backup.
- **-Lib** list of libraries to be compacted, enclosed in quotations. Default is all libraries.
- **-Path** library search path enclosed in quotations. Default is current directory.

Alternatively, from within a Cadence design session, select **CIW→File→Defragment Data→Library**...

and complete the form, indicating the name of a library you wish to defragment.

2.9 Creating a New Library and Design

1. Create a new library called *TestLib.*

   In the CIW window, select **File→New→Library...** and enter the following values in the form:

   Name `TestLib`
   Directory (select the directory for your library; the default is your work directory.)

   Under "Technology File", select "Attach to an existing techfile" if it is not already selected. This method is recommended because any updates to the technology library data (e.g. bug fixes made by VRG staff) will be reflected immediately in your own library, and because of disk-space savings: attaching to a library prevents the copying

of many basic cells and verification files into your disk area. The "Compile a new techfile" method is used only if you plan to make your own modifications to library information; this is useful for researchers developing their own technologies, and is not recommended for novices.

2. Click on **OK** to create a new library. If you selected the "Attach" method for creating a new library, you will see a pop-up prompting you to select the technology library to which you wish to attach. In general, select the technology library named after the technology you are using. For ***CMOSP35***, select the ***cmosp35*** technology library, then click on OK. If you selected the "Compile" method for creating a new library, Cadence will prompt you for the full path to the ASCII technology file to be compiled into your library[1]. If a pathname appears in the pop-up, it should be safe to use it. Click "OK". After the library is created, it will be accessible from the Library Browser and Library Manager.

3. Move the cursor into the CIW window and use the **CIW→Tools→Library Manager** pull-down item, or press **F6** to invoke the Library Manager.

4. Click on *TestLib* in the Library Browser. If you selected the "Attach" method, you will see no cells in your new library, but the basic cells for the technology will still be accessible from the technology library to which your library is attached. If you selected the "Compile" method of library creation, you will see a list of cells inside the library; these cells are used for wiring and for the place and route process.

5. Next, create a new cellview: Select **CIW→File→New→Cellview...**

6. In the new pop-up window, select your library name: click-and-hold on the button beside on the button beside "Library Name", and move the cursor until your library is highlighted, then release the mouse button. Type a new cell name (e.g. MyDesign). Type the view name "schematic" or use the Tool selection method to choose "Composer - Schematic" to automatically fill in the View Name box. (This latter method is preferred, because view names must be typed properly; a typing mistake in the view name could cause unexpected behavior during design sessions.) For other view types (e.g. layout or symbol), select an appropriate entry in the View Name box.

2.10 Exiting from a Cadence Design Session

To exit from a Cadence design session, select **CIW→File→Exit** and then click on **Yes** when prompted *OK to exit icfb?* If you see a pop-up asking if you wish to save your display properties, it is safe to select Cancel, at which time your Cadence session will end. If, however, you see a pop-up asking if you wish to save your work, then Cadence has noticed that you have neglected to save some of your work. You may save all of the unsaved cells (to commit the designs to disk), none of them (to discard any changes), or you may select the cells to be saved before Cadence exits by turning on specific check-boxes.

2.11 Printing Schematic or Layout Views

1. Open your schematic by selecting the cellview in the Library Manager.

1. Each system technology library will have its own version of technology file compiled into it. For CMOSP35, the technology file is located in */CMC/kits/cmosp35/cmosp35.tf*.

2. Select **Design→Plot→Submit...**. The Submit Plot window appears.
   a. In the "Plot" section, verify that the cellview information is correct.
   b. In the "Plot With" section, turn off the Header option.
   c. You can safely ignore the "Template File".
   d. The plotter name and paper size have default values. If these are not appropriate ...

3. Click on Plot Options... to pop up the Plot Options form. You have a choice of plotter name. To print to the Hewlett-Packard LaserJet in LP477, select lj477.

   a. If you don't want Cadence to mail you a plot notification message, you can also turn off the "Mail Log To" option at the bottom of the form.
   b. Click on OK to exit the options form.

4. Click on OK in the Submit Plot form. A message will be displayed in the CIW regarding success or failure.

5. If you want to obtain a plot file in Encapsulated PostScript format, suitable for importing into FrameMaker or other document preparation software, select the plotter name EPS, and indicate that the design is to be plotted to a file. Type the name of the file into which the EPS data will be saved.

2.11.1 Adding your own plotter configuration

After loading default plotter initialization data from `/CMC/kits/VRGlocal/cdsplotinit.vrg` the Cadence tools will read additional configuration data from the file `$HOME/.cdsplotinit` if it exists. You can customize your Cadence plotting needs by preparing this file. For details, see on-line Cadence documentation in **cdsdoc**. EECG users need to copy the file `/CMC/kits/VRGlocal/cdsplotinit.eecg` into their own `$HOME/.cdsplotinit` to enable plotting.

2.12 Cadence Session Logs

A record of each of your Cadence design sessions -- all of the messages appearing in the CIW -- is saved in a file in the directory `$HOME/CDSlogs/` and is named `CDS.log.<process_ID>`. In theory, you can use a log file to recreate a previous session, but you might need assistance from VRG staff. In practice, these log files are only useful for VRG staff to help debug problems you might be experiencing in your design sessions. These log files could be very large, depending on the duration of your design sessions. It is suggested that you clear the CDSlogs directory at least once a week depending on how frequently you use Cadence. An example of commands you can add to your $HOME/`.logout` file to do this automatically when you logout is available in the file `/CMC/kits/VRGlocal/logout_sample`

2.13 CMOSP35 Specific Items

2.13.1 Design Kit Libraries

Several libraries are provided with the current release of the CMOSP35 Design Kit:

Library Name	Description
cmosp35	Cells for schematic capture, layout extraction, netlisting and LVS
padsp35	I/O pads and corner cells
padsp35_dsm	I/O pads and corner cells to be used with Place-and-Route
tcb773p	Black-box standard cells
tpd773pn	Black-box 3.3V I/O cells
tpz773pn	Black-box 5V I/O cells
wcells	I/O and Standard Cells created by CMC.

2.13.2 Transistor models

MOS transistor models are available for Synopsys HSPICE and Cadence SPECTRE. These circuit simulators can automatically select valid models for transistors if the lengths and widths fall within the following valid ranges, when you specify model names "nch" and "pch" for n-channel and p-channel devices, respectively.

Model name	length range		width range		unit
	minimum	maximum	minimum	maximum	
nch.1	1.207	20.0	1.183	19.983	um
nch.2	0.807	1.207	1.183	19.983	um
nch.3	0.35	0.807	1.183	19.983	um
nch.4	1.207	20.0	0.783	1.183	um
nch.5	0.807	1.207	0.783	1.183	um
nch.6	0.35	0.807	0.783	1.183	um
nch.7	1.207	20.0	0.4	0.783	um
nch.8	0.807	1.207	0.4	0.783	um
nch.9	0.35	0.807	0.4	0.783	um
nch.10	1.207	20.0	19.983	200.0	um
nch.11	0.807	1.207	19.983	200.0	um
nch.12	0.35	0.807	19.983	200.0	um
pch.1	1.2228	20.0	1.1823	19.9823	um
pch.2	0.8228	1.2228	1.1823	19.9823	um
pch.3	0.35	0.8228	1.1823	19.9823	um
pch.4	1.2228	20.0	0.7823	1.1823	um
pch.5	0.8228	1.2228	0.7823	1.1823	um
pch.6	0.35	0.8228	0.7823	1.1823	um
pch.7	1.2228	20.0	0.4	0.7823	um
pch.8	0.8228	1.2228	0.4	0.7823	um
pch.9	0.35	0.8228	0.4	0.7823	um
pch.10	1.2228	20.0	19.9823	200.0	um
pch.11	0.8228	1.2228	19.9823	200.0	um
pch.12	0.35	0.8228	19.9823	200.0	um

2.14 CMOSP18 Specific Items

2.14.1 Design Kit Libraries

Several libraries are provided with the current release of the CMOSP18 Design Kit:

Library Name	Description
cmosp18	Cells for schematic capture, layout extraction, netlisting and LVS
package	Chip housing packages and models of pad and bondwire inductance
tpz973g	Black-box I/O cells
virage_sram	Some pre-compiled memory blocks
vst_n18_sc_tsm_c4	Black-box standard cells

2.14.2 Transistor models

MOS transistor models are available for Synopsys HSPICE and Cadence SPECTRE. These circuit simulators can automatically select valid models for transistors if the lengths and widths fall within the following valid ranges, when you specify model names "nch" and "pch" for n-channel and p-channel devices, respectively. Use the equivalent ranges, but names "nch3" and "pch3" instead, for 3.3-volt n-channel and p-channel devices.

Model name	length range		width range		unit
	minimum	maximum	minimum	maximum	
nch.1	1.2	21.0	10.1	101.0	um
nch.2	0.5	1.2	10.1	101.0	um
nch.3	0.18	0.5	10.1	101.0	um
nch.4	1.2	21.0	1.3	10.1	um
nch.5	0.5	1.2	1.3	10.1	um
nch.6	0.18	0.5	1.3	10.1	um
nch.7	1.2	21.0	0.6	1.3	um
nch.8	0.5	1.2	0.6	1.3	um
nch.9	0.18	0.5	0.6	1.3	um
nch.10	1.2	21.0	0.22	0.6	um
nch.11	0.5	1.2	0.22	0.6	um
nch.12	0.18	0.5	0.22	0.6	um
pch.1	1.2	21.0	10.1	101.0	um
pch.2	0.5	1.2	10.1	101.0	um
pch.3	0.18	0.5	10.1	101.0	um
pch.4	1.2	21.0	1.3	10.1	um
pch.5	0.5	1.2	1.3	10.1	um
pch.6	0.18	0.5	1.3	10.1	um
pch.7	1.2	21.0	0.6	1.3	um
pch.8	0.5	1.2	0.6	1.3	um
pch.9	0.18	0.5	0.6	1.3	um
pch.10	1.2	21.0	0.22	0.6	um
pch.11	0.5	1.2	0.22	0.6	um
pch.12	0.18	0.5	0.22	0.6	um

Chapter 3 Cadence: Schematic Entry

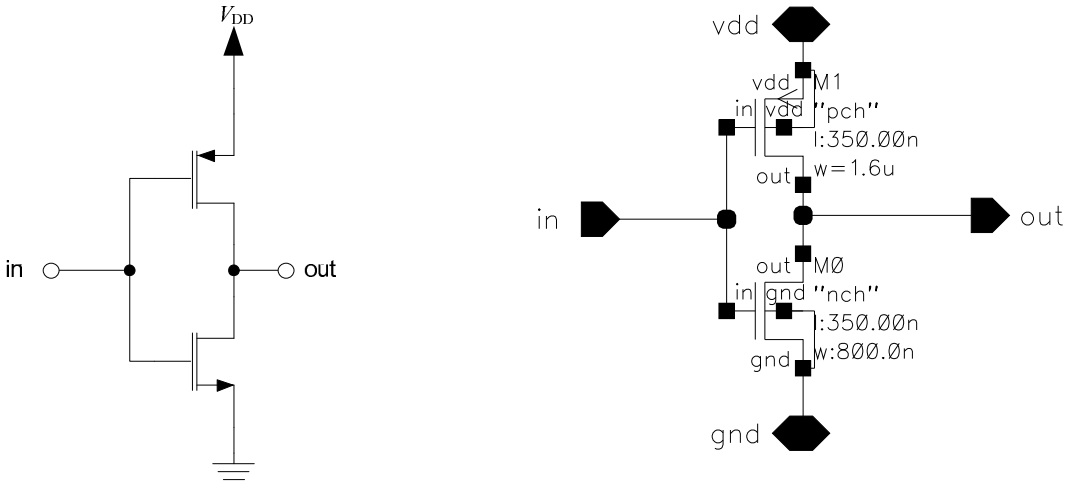
3.1 Power Supply Connections

Some technology-dependent design-kits allow for the use of global nodes in a design. When using the Cadence tools, global nodes are identified by an exclamation point following the node name: *gnd!*, *vdd!* or *vss!* for example. However, some libraries in the CMOSP35 design kit do not allow the use of global node names, but circuit simulators, such as Spice, require that at least a global ground node exists.

To overcome this limitation make sure that ground and power connections are created for each component in your design hierarchy. Then, global power connections can be made at the top level of your design hierarchy, such as in the test bench schematic. Alternatively, you can use global node names exclusively in your design, where supported.

3.2 Creating Your First Transistor-Level Schematic

This section will guide you though the steps needed to take to create your first Cadence schematic, a transistor schematic for a logic inverter as shown in Fig. 3.1. Steps will be provided for both the CMOSP35 and CMOSP18 technology nodes.



(a) inverter schematic **(b) Cadence dfII schematic**

Figure 3.1: Schematic of logic inverter

1. Create a new library, or use the existing library *TestLib* library from the previous chapter, for the technology node with which you wish to work. If creating a new library, use the *Attach to existing techfile* method described in the previous chapter.

2. Create a new cell view in your library, in the CIW window, select **File→New→Cellview**

   The Create New File form appears; fill in the following values:
   Library Name: *TestLib* (or select your library from the Library drop-down selection list)
   Cell Name:*inverter*
   View Name: *schematic* (or select Composer-Schematic from the Tool drop-down selection list)

Note: Make sure that you spell the view name correctly, otherwise the Cadence tools will not recognize the new cellview as a schematic. In that case, tools and menus associated with a schematic will not be accessible.

Click OK to create the new cell. A new schematic design entry window will open with the name of the library, cell, and cellview displayed in the windows title bar. A schematic specific pull-down menu should now be available along with a shortcut icon menu along the left size of the window.

3. Create and place an instance of an n-channel MOS transistor, nfet, in your schematic:
   a. Select **Add→Instance** from the pull-down menu or use the **i** keyboard shortcut key to open the *Add Instance* dialog.
   b. Click the Browse button to bring up the *Library Browser.*
   c. In the *Library Browser* dialog window select the following values or fill in the appropriate fields in the *Add Instance* dialog window:

	CMOSP35	**CMOSP18**
Library	cmosp35	cmosp18
Cell	nfet	nfet
View	symbol	symbol

   d. In the *Add Instance* form specify the channel width and length of the nfet in meters. Dimensions can be specified in either SI units or scientific notation. Enter 800n or 800e-9 to specify the width of the new nfet instance. Keep the default value of 350n for the length of the transistor instance.
   Note: Although there are different models for different transistor size ranges, you don't have to specify which model is to be used. During simulation the appropriate model will be automatically selected based on the device's size. A simulation error will occur if an appropriate transistor model cannot be found.
   e. Without closing the Add Instance dialog, return to the schematic window. Notice that the outline of the transistor symbol follows the mouse cursor. Left-click to place an instance of the nfet symbol. Transistor parameters will be displayed beside the placed symbol.
   Note: Multiple instances of a similarly sized transistor can be placed sequentially, until you abort by pressing the ESCAPE key or Cancel on the *Add Instance* dialog window. You can also change the size, or other parameters, of subsequent instances by bringing up the *Add Instance* dialog.

4. Repeat the above steps in order to place a pfet instance that has a channel width of 1.6um and channel length of 350nm.

   Note: There are four generic transistor symbols: nfet, nfet3, pfet, and pfet3. The nfet3 and pfet3 symbols have the body substrate connection made internally to the source of the device. Whereas, nfet and pfet symbols are the typical 4-terminal transistor devices, which allow you to explicitly specify the substrate connection.

   Note: You can edit the properties of any symbol after it has been placed, by selecting the symbol and using the q keyboard shortcut key. This will bring up the Edit Object Properties dialog which allows you to modify that device's parameters.

5. Place schematic pins for the input, output, and power supplies connections:

   a. Select **Add→Pin** or use the **p** keyboard shortcut key.
   b. In the **Add→Pin** dialog enter the names for the input, output and power supply terminals in the Pin Names field: **in out vdd gnd**. This will allow you to place these pins in a sequential manner, starting with **in**; pin names will vanish from the Pin Names field as you place them.
   c. Before placing the pins specified in the Pin Names field, make sure to select the proper direction for each pin:

Pin Name	Pin Direction
in	input
out	output
vdd, vss	inputOutput

   Note: Pin naming convention is not fixed across all of the CAD tools you might use. However, it is generally safe to assume that all tools will at least support alphanumeric and numeric pin names, as long as the first character is not a number. Punctuation characters could be problematic, so it is best to avoid these.

   Note: HSpice is NOT case sensitive. Do not use case to differentiate between two distinct nets.

6. Zoom out so that you can see all of the symbols you have placed, select **Window→Fit**

7. Connect symbols and pins using wires:

   a. Select **Add→Wire (narrow)** or use the **w** keyboard shortcut key.
   b. Make the required connections, as shown in Fig. 3.1, by left-clicking for each end-point and turning point of the interconnecting wire; double-clicking will terminate the wire being currently drawn.

8. You can also connect different symbol or pin terminals by using common wire names; wires with identical names are considered to be implicitly connected. This might greatly simplify your schematic and make it easier to understand by greatly reducing the amount of wiring in the schematic.

   a. Draw a wire segment from a symbol or pin terminal.
   b. Select **Add→Wire Name** or the **l** keyboard shortcut key to bring up the *Add Wire Name* dialog window.
   c. Enter the names of the net names in the Names field; net names will vanish from the Names field as you assign labels to wires.
   d. Place the labels by clicking over the desired wire.

9. Run a schematic check: select **Check→Current Cellview**. Check the CIW window for any schematic errors or warnings identified. A detailed description of the schematic check is given in Section 3.6.

10. Save your design by selecting **Design→Save**, using the **S** keyboard shortcut key. You can also use the first icon at the left hand side of the schematic window, **Design→Check and save**, or the **X** keyboard shortcut key to perform an automatic schematic check and save.

11. Plot your schematic: select **Design→Plot Submit**. In the *Submit Plot* dialog window select an appropriate printer and plot size by following the **Plot Options** button at the bottom of the window.

3.3 Creating a Schematic Symbol

For hierarchical designs, smaller components are used to build up successively bigger designs. To be able to use the Inverter design as part of another schematic it is necessary to first create a symbol view of the schematic.

> **Note:** You can also use this methodology to create a symbol view from an extracted layout cellview, later on during physical design.

1. Open the Inverter schematic.

2. Select **Design→Create Cellview→From Cellview...**

3. In the *Cellview from Cellview* dialog window keep the default settings, and click OK.

4. A *Symbol Generation Options* dialog window will pop-up. This window allows you to choose the positioning of the various pins on the four sides of the symbol, ordered top-to-bottom and left-to-right.

   Note: It is good practice to establish a pin ordering convention for all symbols. A typical convention is to put all inputs on the left side of the symbol, all outputs on the right, low voltage (ground) supply connections on the bottom, and high voltage (VDD) connections on the top.

5. Click the **Load/Save** selection box; in the expanded area of the dialog box select analog from the *Load Symbol Template Configuration* drop-down list and click the **Load** button. This will ensure that the proper symbol template is loaded for use with the Analog Design Environment, the Cadence simulation GUI.

6. Click **OK**. The newly created symbol cellview will be displayed.

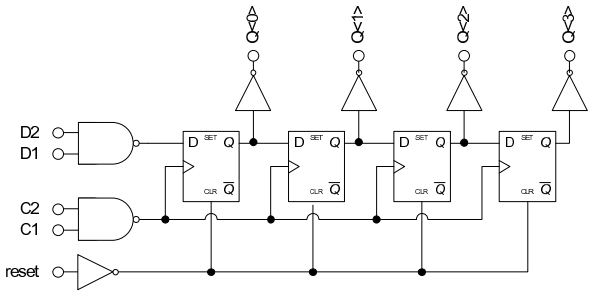
3.4 Creating a Hierarchical Schematic

1. Create a new schematic cellview named *shift* in your library and then edit the *shift schematic*.

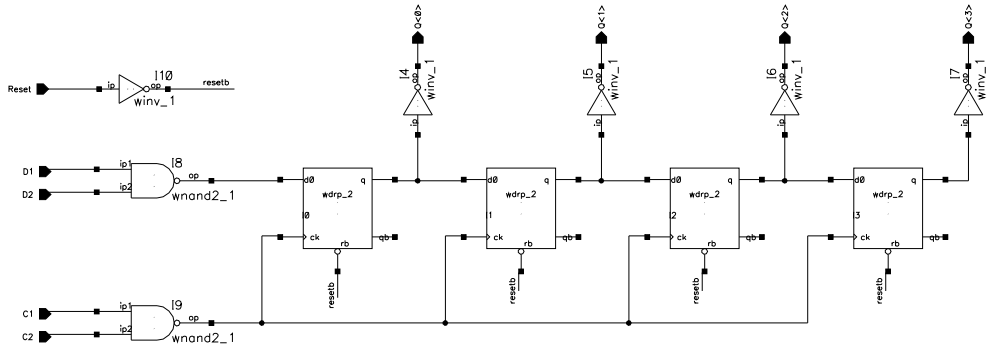
2. Place an array of flip-flops in your schematic, as shown in Fig. 3.2.

   a. Select **Add→Instance** to open the *Add Instance* dialog window.
   b. Select an instance of a positive clock-edge triggered D flip-flop with 2 drive strength: Using the *Library Browser* select

	CMOSP35	**CMOSP18**
Library	wcells	vst_n18_sc_tsm_c4
Cell	wdrp_2	DEFPQ2
View	symbol	symbol



(a) shift schematic



(b) Cadence dfII schematic

Figure 3.2: Schematic of four flip-flop shift circuit

c. In the *Add Instance* dialog window specify the size of the array at the bottom of the form to be Rows: 1, Columns: 4.
d. Place the first instance of the flip-flop.
e. After you have placed the first instance, you should see outlines of all other instances in the array move along with your mouse cursor. The second instance placement indicates the spacing to be used for the second and successive instances. Place the second cell in order to automatically place a horizontal row of flip-flops.
f. Click Cancel or press the ESC key to close the *Add Instance* dialog window.

3. Place an array of inverters, you will need to click Rotate in the Add Instance dialog window in order to orient the inverters properly:

	CMOSP35	**CMOSP18**
Library	wcells	artisan_sc_30
Cell	winv_1	INVX1
View	symbol	symbol

4. Place the rest of the gate instances as shown in Fig. 3.2 using the following symbol for the NAND gates.

	CMOSP35	**CMOSP18**
Library	wcells	artisan_sc_30
Cell	wnand2_1	NAND2X1
View	symbol	symbol

5. Add input and output pins to your design. Bring up the Add Pin dialog window and enter the following pin names: **D1**, **D2**, **C1**, **C2**, **Reset**, **Q<0:3>**.

   Note: A bus is defined by labeling a wire with the bus name followed by the bit range of the bus: *busName<LSB:MSB>*. It is common practice to use a thick wire to distinguish a bus from a wire. To bring out a single wire from the bus, connect a wire to the bus and label it specifying the bus name and bit number in angle brackets: for example *busName<bitNumber>*.

6. Interconnect the gates and pins together, as shown in Fig. 3.2, using wires by selecting **Add→Wire (narrow)** from the pull-down menu, or by using the **w** keyboard shortcut key.

   Note: If you want to place an array of wires, place one wire, select it, and then choose **Edit→Copy** from the pull-down menu or use the **c** keyboard shortcut key. Use the array option in the *Copy* dialog window.
   a. Select input as the pin direction and place the input pins in the schematic.
   b. Select output as the pin direction and select Bus Expansion, Multiple Placement, and click Rotate and place the output pins.
   Note: You will also need to create pins for the power connections if you are not using global net names for the power supplies.

7. Run a design check to establish hierarchical connectivity of your design and to add that information into the design's database. Select **Check→Current Cellview** and check the CIW for any errors and warnings. You should get four warnings related to the floating **qb** output pins for each of the four flip-flops. (You may ignore these.)

   Note: You can use **Check→Find Markers** to find and explain error or warning markers in the schematic.

8. Save your design.

3.5 Traversing the Design Hierarchy

When working with a hierarchical schematic you can always view the details of instance cells by using the **Descend** command. In the case of design kit cells, schematic views may not be available, but you can descend down in the hierarchy to see the cell's other available views, such as layout, extracted, etc.

1. Select an instance of the flip-flop cell. Press and hold the middle mouse button to invoke the Instance pop-up menu and select *Descend Read...* or use the **e** keyboard shortcut.

2. In the *Descend* pop-up dialog, select layout in the *View Name* drop down list and click **OK**. The mask layout for the flip-flop cell should now be displayed.

   Note: To view all mask detail contained within sub-cells of this layout, use the **F** keyboard shortcut. To turn off this detail use the **CTRL+F** keyboard shortcut.

3. To go back a level in the hierarchy select **Design→Hierarchy→Return** or use the **B** keyboard shortcut.

3.6 Running a Schematic Check

This feature runs some simple tests to ensure that a schematic you have drawn has no major errors, like floating wires, unconnected pins, shorted output pins, and others.

1. Open your *TestLib shift schematic* for editing.

2. Initially we need to examine and edit the rules used to perform the schematic check. Select **Check→Rules Setup**. The *Setup Schematic Rules Checks* pop-up dialog window will appear.

   a. Select **Normal** from the *Packaged Checks* drop-down list.
   b. In the *Logical* tab, select the **warning** radio button for the *Floating Output pins* check rule.

3. Run a design check on the schematic by selecting **Check→Current Cellview** or use the **x** keyboard shortcut.

4. The resulting pop-up dialog window summarizes the number of errors and warnings found in your schematic. The errors and warning will also be highlighted on the schematic. The CIW window will list the warnings and errors generated in detail. The following is a list of warnings that you should get for the TestLib shift schematic.

```
Extracting "shift schematic"
Warning: Pin "qb" on instance "I3": floating output.
Warning: Pin "qb" on instance "I2": floating output.
Warning: Pin "qb" on instance "I1": floating output.
Warning: Pin "qb" on instance "I0": floating output.
```

5. You can also step through all warnings and errors one at a time with visual feedback on the schematic. Select **Check→Find Marker**... or use the **g** keyboard shortcut. The *Find Marker* pop-up dialog window will appear. Select **Zoom to Markers** in the dialog window to have the display zoom in (very closely) to the error/warning reported in the list.

6. Once you have verified that a set of warning/errors are acceptable, you can disable them from being checked by future checks by modifying the *Check Rules*. It is possible to get multiple warnings/errors for one problem, so re-run the check after repairing problems to ensure that all errors are repaired.

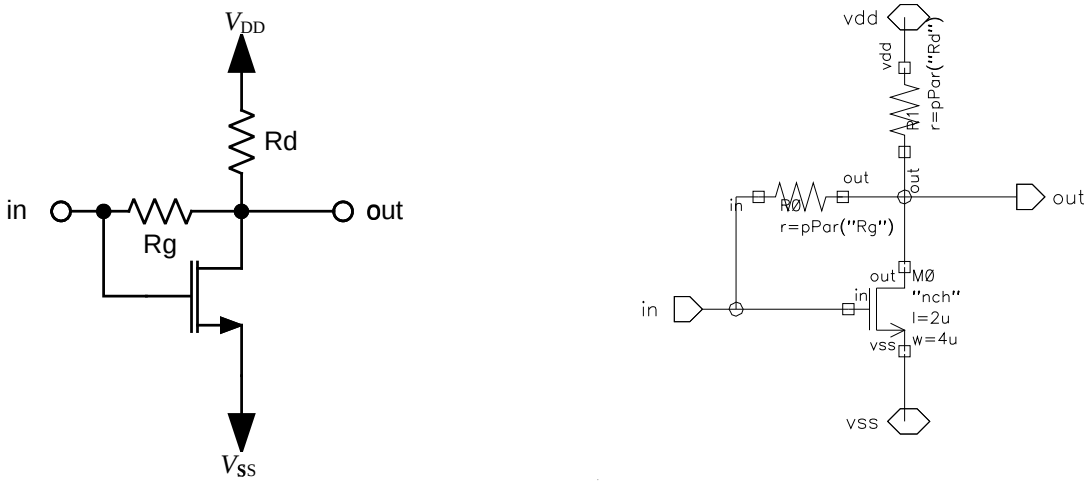
3.7 Pass Parameters and Variables

Quite often, a circuit structure is repeatedly in the top level design. Cadence Analog Artist, the circuit simulator interface, has an efficient way to handle multiple uses of a circuit structure where the difference in

each occurrence is the value of one, or more, of its basic components (such as resistor values, transistor sizes, etc.). Cadence allows you to assign basic component values as pass parameters in the building block and lets you pass the desired component parameters to each block (in the form of a user-defined symbol) from a higher hierarchy level. This is analogous to a software program, where a function may be defined once, but called from a higher level function numerous times and with different parameters each time.

Another method to vary component values is through the use of component variables. This is used at the top-level design for parametric analysis. Parametric analysis allows you to sweep specific component values during a schematic simulation. This is described later in a chapter on schematic simulation. In the following example, both pass parameters and top-level component variables are used in a design.

1. Create a new amplifier schematic in your TestLib library. Enter the schematic as shown in Fig. 3.3. Use components *resisto*r and *nfet3* from the *cmosp35* or *cmosp18* library, depending on which technology node you are working with.



(a) amplifier schematic (b) Cadence dfII schematic

Figure 3.3: Schematic of CMOS amplifier

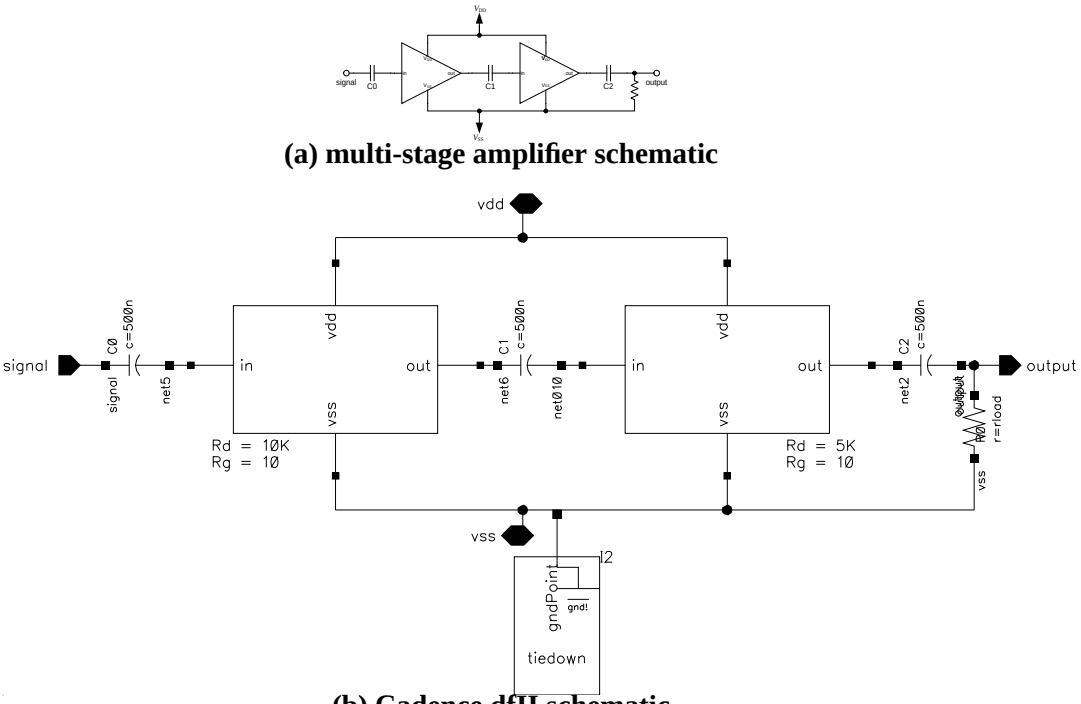
2. Assign pass parameters for the resistor values: instead of typing numeric values in the *Add Instance* pop-up dialog window, use the following resistor ``values": **pPar(``Rd.")** and **pPar(``Rg")**, for the drain and gate resistors respectively.

3. Create a symbol for the amplifier schematic.

4. Create a new *mul_stage schematic*. Enter the schematic as shown in Fig. 3.4.

5. When you instantiate the amplifier symbol, you will find two CDF (*Component Description Format*) parameters - Rd and Rg - at the bottom of the Add Instance pop-up dialog. Enter the values as shown.

6. When placing the resistor, assign *rload* as the resistance value. This is now a component variable, which may be assigned a real value during simulation. (This topic will be discussed in a later chapter.)



(a) multi-stage amplifier schematic

(b) Cadence dfII schematic

Figure 3.4: Schematic of a multi-stage hierarchical CMOS amplifier

7. Add wires to connect the *vdd* and *vss* terminals of your amplifier symbols.

8. Check and save the schematic.

3.8 Schematic Entry Keyboard Shortcuts, or *bindkeys*

Command	Bindkey	Menu Selection
Save	S	Design→Save
Check & Save	X	Design→Check and Save
Check Current Cellview	x	Check→Current Cellview
Find Markers	g	Check→Find Marker...
Add symbol instance	i	Add→Instance
Edit Object Properties	q	Edit→Properties→Objects...
Add pin	p	Add→Pin
Add wire	w	Add→Wire (narrow)
Label wire	l	All→Wire Name
Zoom to Fit	f	Window→Fit
Zoom to selection	z	Window→Zoom→Zoom In
Zoom out 2x	[	Window→Zoom→Zoom Out by 2
Zoom in 2x	]	Window→Zoom→Zoom In by 2
Descend Hierarchy (read)	e	Design→Hierarchy→Descend Read
Descend Hierarchy (edit)	E	Design→Hierarchy→Descend Edit
Ascend Hierarchy	B	Design→Hierarchy→Return

Chapter 4 Cadence: Physical Layout

The layout view of a design consists of the actual mask features (rectangles and polygons) which are to appear on the final masks used during the fabrication of a microchip. Cadence Virtuoso, the layout editor, allows each mask layer to be represented on the screen by a different colour or stipple pattern. These layer patterns are indicated in the Layer Selection Window (LSW) during a layout session. In this window, the *current layer* (i.e. the layer in which any new feature will be drawn) is highlighted. This current layer can be changed by left-clicking on the layer name in the LSW.

> **Note:** There are several copies of some layers in the LSW. Normal drawing layers are denoted by a layer type of *dg*, for drawing. There are other types of layers as well, each layer type having a special purpose, such as *pn* for pins, *nt* for nets, *wg* for warnings, and *er* for errors. Designers should use only the *dg* layer to draw mask features.

4.1 Creating a Mask Layout

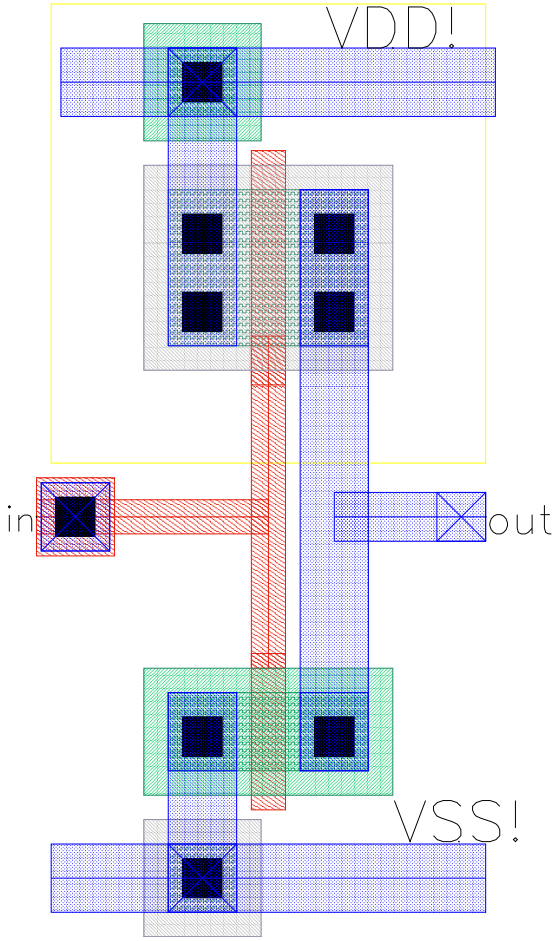


Figure 4.1: Example layout of a logic inverter

1. Create a new inverter layout cell view in your TestLib library: **File→New→Cellview**

   Library Name: **TestLib** or select your library from the Library drop-down selection list
   Cell Name: **inverter**
   View Name: **layout** or select Virtuoso from the Tool drop-down selection list

2. Fig. 4.1 shows one possible layout of a CMOS inverter. Try to reproduce this layout. The PMOS transistor has a W/L of 1.6µm / 0.35µm, while the NMOS has a W/L of 0.8µm / 0.35µm.

   The PMOS transistor consists of a polysilicon gate crossing active covered by p-plus in an n-well. The NMOS transistor consists of polysilicon gate crossing active covered by n-plus. Note also the n-well contacts (active covered by n-plus in the n-well) connected by metal to VDD near the top of this layout, and the substrate contacts (active covered by p-plus outside the n-well) connected by metal to VSS near the bottom of the layout. These represent the *bulk* connections for the MOS transistors.
   Mask features are drawn in the layout editor by:
   - left-clicking on a drawing layer in the LSW to make it the current layer
   - invoking one of the **Create→** commands (rectangle, polygon, or path)
   - for a rectangle, left-clicking the diagonally opposite corners, or,
   - for a polygon, left-clicking the polygon points in sequence, with a double-fast-click on the final point.

   Note: Any polygons drawn on the same layer that touch or overlap are considered to be electrically connected.

   Alternatively, you can use user customizable template MOS transistor footprints from the *pcell* or *CMCpcells* libraries, depending on your technology:

   a. Select **Create→Instance** or use the **i** keyboard shortcut, which brings up the *Create Instance* pop-up dialog window.
   b. Select the **pcells** library for the *CMOSP35* technology node or the **CMCpcells** library for the *CMOSP18* technology node. You now have several templates to choose from, described briefly below.

CMOSP35		**CMOSP18**	
nfet	NMOS template, no bulk connection	**spcnmos**	Customizable NMOS template: optional gate splitting, source, drain, or substrate contacts, and substrate well
nfet3	NMOS template, bulk shorted to source		
pfet	PMOS template, no bulk connection	**spcpmos**	Customizable PMOS template: optional gate splitting, source, drain, or substrate contacts, and substrate well
pfet3	PMOS template, bulk shorted to source		

   c. Specify the width and length of the gate, as well as any other required options and place the transistor instance.
   Note: You might not be able to see the transistor instance details; if this is the case, then use **SHIFT+f** to show instance details.

4.1.1 Making connections using Path Stitching

It is possible, although extremely tedious, to connect polysilicon or metal connections on transistor together using polygon shapes for wires, vias, etc. However, a much easier alternative is to draw paths, which can be made to traverse several layers as the connection crosses over or under existing objects in the layout.

When drawing paths, it is very important to use a path whose width is an even multiple of design grid units. Since a path is defined by a set of coordinates along its center-line (with half of the path-width on either side), an odd number of snapping grid units in width will result in the edges of the path being off-grid (that is, on a 0.05 grid in the CMOSP35 technology). This will not be flagged as a design rule check (DRC) violation in Cadence Diva (the layout verification tool), but may result in DRC violations when a design is processed at CMC for fabrication using DRACULA, another verification tool. A similar result may occur for 45-degree geometries, therefore you are encouraged to use Manhattan geometry if possible.

When a design is submitted for fabrication to CMC, all paths are converted to polygons. The ends of each path will be converted into round tips. If you have a path butted to another geometry, the converted geometries will cause DRACULA DRC violations. There are two methods to avoid the violations: 1) extend the end of the path into the connecting geometry, 2) convert all paths into polygons in Cadence (Cadence will create squared ends for the converted wires).

In some cases, you cannot draw a path on a single layer directly between two points due to intervening mask features, so you use path stitching to change layers within the path. Path stitching automatically changes the path from one layer to another, placing an appropriate contact to connect the two layers. The Path command selects the contact from the library technology file.

1. Turn on gravity to snap the cursor to objects
   a. Press **SHIFT+f** to display all levels.
   b. Select **Options→Layout Editor...** which brings up the *Layout Editor Options* pop-up dialog window.
   c. Check to make sure *Gravity* is selected.
   d. Change *Aperture* to **2**. The gravity aperture controls the distance at which the cursor snaps to objects: a setting of 2 means that the cursor should snap to an object when it is within 2 to that object.
   e. Change *Depth* to **2**. The cursor can snap to object within instances. The depth setting specifies how many layers down in the hierarchy the cursor should snap to.
   f. Click **OK**.

2. To draw a path using the *metal1* layer, click on the *metal1 dg* layer in the LSW.

3. Select **Create→Path**, bringing up the *Create Path* popup dialog window.

   Note: The path width is set to 0.5μm, which is defined in the technology file as a property of the metal1 layer.

4. Select **Design→Options→Display Options** and set *Snap Modes* to orthogonal.

5. Start making the connections between objects by clicking on an appropriate cell layer to start the path. Left click on a point where you'd like to change path directions.

6. Use the *Change To* field in the *Create Path* popup dialog window to change from *metal1* to *metal2*. Press *metal1* next to *Change To* in the *Create Path* popup dialog window. A list of layer names appears; these are the layers you can change to from metal1, based on the metal1 contacts defined in the library's technology file.

7. Slide the cursor to *metal2* and release the mouse button.

8. Move the cursor back into the cell view. A metal1 to metal2 contact appears on top of your cursor.

9. Click to anchor the contact.

   Note: The width of the *metal2* path changes to 0.6µm, as defined in the technology file.

10. You can change to any of the given layers during path stitching. Double click on the same point to complete the path currently being drawn.

11. Complete the connections of the entire layout.

4.1.2 Creating Labels

Labels are a handy way of embedding more information in a layout, in such a way that will not affect the resulting layout. Labels serve the same purpose in layouts as comments do in software code.

1. Check the LSW for *text dg* (text drawing layer), *metal1 pn* (metal1 pin layer) and *metal2 pn* (metal2 pin layer). If these layers do not exist, you have to include them in the list of valid layers that are shown in the LSW.

   In the LSW, select **Edit→Set Valid Layers**. A popup window with all layers defined in the technology file is displayed. Look for the above layers and select them by clicking on the box after the layer name.

2. Create labels for information purposes, by clicking on the *text dg* layer in the LSW.

3. Select **Create→Label**, the *Label* popup dialog window will appear.

4. Type **vdd** in the *Label* field. Move the cursor over the power line at the top of the cell. Click to place the label.

5. Create labels for **vss**, **in**, and **out**.

4.1.3 Creating Pins

Pins define the terminals that can be connected to during hierarchical layout or automatic place and route. Pins, as well, can serve to be the starting point for an automatic layout vs. schematic (LVS) comparison.

1. Click on the *metal1 pn* layer in the LSW. The pin must be drawn on the same layer as the underlying wire. The only difference is that the *pn* layer must be used for pins, instead of the *dg* layer.

2. Select **Create→Pin**, which will bring up the *Create Pin* popup dialog window.

3. Type **VDD! VSS!** in out as the terminal names.

   Note: You can specify any number of terminal names; Each pin you draw is assigned the next name in the *Terminal Names* field.

4. Set the *Access Direction* field accordingly. For instance, *VDD!* and *VSS!* will have access from left and right only. (For more explanation in access direction, refer to the on-line Cadence manuals). Access direction is only necessary if you plan on using automatic routing.

5. Set the *I/O Type* to **inputOutput** for the *VDD!* and *VSS!* pins. Specify the *I/O Type* to **input** and **output** for the *in* and *out* pins, respectively. You might also need to change the *Access Direction* appropriately for the *in* and *out* pins.

6. Draw the rectangle for the *VDD!* pin coincident with the power line at the top of the inverter. The name *VDD!* disappears from the *Create Pin* dialog window. The next name at the beginning of the list is *VSS!*.

7. Press **ESC** to stop drawing pins and save your layout.

4.1.4 Hierarchical Layouts

Designs can typically be broken down into some kind of hierarchy, with the goal of having some basic leaf cells, and possibly nodes, repeated several times in the hierarchy. Such an approach tends to be much more efficient in terms of human layout effort and in disk size in terms of overall design. Placing instances of sub-cells may be accomplished in much the same way as described previously for schematic designs.

As an exercise, try creating an inverter chain. You may wish to modify your initial inverter layout such that the input and output of the inverter cell are on the same metal layer, which will minimize the amount of interconnect required between instances. Typically, designers will specifically engineer a cell's layout, so that some connections can be automatically made when different or similar cells are abutted against each other. Such an approach is especially useful for power rail connections and in general minimizes the amount of manual connection routing effort.

Layout instances of a cell can be made using **Create→Instance** or by using the **i** keyboard shortcut. Specify the *Library* and *Cell* that you wish to instantiate, and make sure that the *View* is set to be **layout**. Place instances in your layout by simply selecting where you'd like them to be placed. You can rotate, horizontally flip, or vertically flip the instance before you place it.

4.1.5 Display Level Control in Hierarchical Layouts

To change the amount of detail shown in a cell's view, you can specify the range of levels in the hierarchy that are to be visible. By default, only drawing layers at the current level in the hierarchy are shown, that is level 0. Any cell instances are level 1, and cell instances inside these instances are level 2, and so on.

To change the levels of detail shown,

1. Zoom out to fit the whole layout in your layout window, using **Window→Fi**t or use the **f** keyboard shortcut.

2. Select **Design→Options→Display**, which will bring up the *Display Options* popup dialog window.

3. Change the *Display Levels* fields so that it will display layers 0 to 1. Click **Apply**.

   Note: Choosing to show levels 0 through 20, will display all levels of the hierarchy. Two keyboard shortcut keys allow you to quickly toggle from displaying level 0 only and levels 0 through 20; these shortcut keys are **SHIFT+f** and **CTRL+f**, respectively.

4.2 Design Rule Check (DRC)

Design rules are specified by foundries for each technology node; these rules roughly specify minimum dimensions and spacings of layout polygons in order to guarantee a desired manufactured die yield. Violation of design rules means that the probability of your design being manufactured correctly decreases. Indeed, some black box logic cells, which you can use for your logic designs, do have design rule violations, but these violations have been shown to achieve a desirable working die yield.

Note: Some foundries or fabrication partners, such as CMC, do not allow you to violate design rules. In fact, CMC will only let you violate design rules if each design rule violation is explained in detail.

To run a DRC,

1. Open the layout view of the design you wish to verify.

2. Select **Verify→DRC**, the *DRC* popup dialog window will appear.

3. Use the default values in the form and click **OK**.

   During the DRC run, the CIW will display status messages indicating the design rule currently being evaluated. Upon completion of the DRC, errors in the design are indicated on the layout via markers.
   The text output in CIW indicates the violated design rules, if any, in addition to the highlighted errors in the layout.

4. To check which design rule is violated on the layout, select **Verify→Markers→Explain**. Point and click on the error marker you wish explained.

5. To check all DRC violations in sequence, select **Verify→Markers→Find** and select the *Zoom to Markers* option in the dialog window that pops up. This method allows you to use Cadence to find and zoom to each DRC error in sequence, which is useful if your layout happens to be large!

6. Correct all the errors until the layout passes DRC.

   Note: To improve the speed of the DRC run, deselect the *Echo Commands* option in the *DRC* dialog window. This will disable the printing of DRC status messages to the CIW window and to the session log file. A summary outlining the results of the DRC run will still appear.

4.3 Layout Extraction

Layout extraction, also known as circuit extraction, is a procedure by which mask features drawn by a designer are automatically recognized by the CAD tool in order to automatically generate an equivalent circuit schematic from a layout. An extracted layout can be used to generate a circuit netlist for simulation purposes. Furthermore, parasitic capacitances and resistances due to the way transistors and other polygons are drawn can be optionally calculated during the layout extraction process.

To make layout extraction possible, technology-specific rules are supplied in order to define the devices and the interconnections between them: i.e. ``polysilicon over active covered by p-plus'' may be recognized as an n-channel MOSFET in the CMOSP35 technology. Transistor device sizes, i.e. the transistors' W/L ratios, are directly calculated from recognized overlapping regions.

To extract a design,

1. Open the layout to be extracted.

2. Select **Verify→Extract**, the *Extractor* dialog window will appear.

   There is one optional switch which may be set for the CMOSP35 technology, parasitics, which will calculate the parasitic capacitance due to all drawn features.
   If no switch name is specified, the layout will be extracted without parasitics. This is sufficient for preliminary verification; during later stages of design, adding parasitics to the extraction will provide a more accurate netlist; hence, a more accurate sign-off simulation.

3. Use the defaults in the form and click **OK**.

4. Cadence will indicate to you the extraction errors, if any, directly on the layout and in the CIW.

5. To check what rules are violated, use the same steps you used as when checking for DRC errors, **Verify→Markers→Find...**

6. Make necessary corrections on the layout and re-extract, repeating until no errors are reported.

   Note: To improve the speed of the extraction run, deselect the *Echo Commands* box in the *Extractor* dialog window. This will disable the print of status messages to the CIW, except for the final summary of violations.

7. You should also run a script to remove any floating nodes from the extracted layout; floating nodes will cause errors in Hspice and Spectre circuit simulations as they do not have a DC path to ground. Select **CMC Skill→Extract→Clean extracted view** and check the CIW for any errors or warnings.

   Note: DRC and extraction may also be done in batch mode for large designs. See on-line documentation or consult VRG staff for details.

4.3.1 Macro Layout Extraction[1]

For designs with a large number of transistors, such as memories or custom digital chips, the extraction method described in the previous section (called flat extraction) can take several hours. A macro extraction methodology can reduce the extraction time to several seconds.

The macro extraction process looks to identify cells that have already been extracted flat and does not re-extract these cells. For example, we can create a layout for a 5-inverter ring oscillator by first creating the layout of a simple inverter and extracting it flat. Then we can place five separate instances of the inverter layout in a new (ring oscillator) layout cell view, add appropriate metal connections, and perform a macro extraction. The specific steps for this 5-inverter ring oscillator are as follows:

1. Create a layout for the basic inverter.

2. Add shape pins on all the terminals. Create the shape pins such that they cover as much of the metal as possible. This part is critical since during macro extraction, only the portions that the pin layers covers are considered as connection points. Multiple pins with the same name can be placed as long as they are electrically connected. Metal or poly that is left uncovered by a pin can potentially cause a short that will go *undetected* by the macro extraction method. Since internal nodes are not covered by a pin, there is always the possibility for an undetected short circuit when using macro extraction. Nevertheless, macro extraction is still useful for quickly extracting large cells.

3. Perform flat extraction on the basic CMOS inverter cell as described in the previous section.

4. Open the newly created extracted view of the inverter cell. Note the pin layer and the extracted metal. Close the extracted view.

5. Create a layout for a 5-inverter ring oscillator by placing five instances of the base inverter cell. Add metal to connect the base inverters in the proper configuration. Carefully add pins on all terminals as in step 2

6. In the layout cell view select **Verify→Extract**; the *Extractor* form appears. Choose **macro cell** as the *Extract Method* and click on **OK**

7. Cadence will report the extraction errors, if any, in the layout cell view and in the CIW; examine and fix the errors if necessary.

8. In the CIW, Cadence will report the number of macro cells it finds. Make sure that Cadence finds the number of macro cells that you expect. In this case we expect Cadence to find five macro cells.

9. Open the newly created extracted view of the ring oscillator. Identify the basic CMOS inverter cells and the metal connections. Close the extracted view.

10. The ring oscillator can now be used in other layout cell views with macro extraction.

1. Special thanks to Kostas Pagiamtzis for preparing this section

If you are using macro extraction, it is a good idea to test it with small test layouts before applying it to a large design. If metal is left uncovered by a pin, there is a possibility that a connection/short can be made in the layout that is not detected by the macro extraction. Thus it is recommend that macro extraction is used as a way to speed up design and development, but final verification should always be performed with a full flat extraction of a design.

When performing a macro extraction it is possible to mark some instances as *flat*. Thus the extractor will descend one level into these cells. To identify an instance as *flat*, select the instance in the layout cell view, select **Edit→Properties**, and add the Property ``*ivCellType*"with value *graphic*. The *graphic property* can also be set in the layout cell view and will affect all instances of the cell.

The *graphic property* only flattens a single level. Thus, to fully flatten a cell with many levels of hierarchy below it, all lower-level cells must have their *ivCellType* set to *graphic*.

4.3.2 Join Nets With Same Name

One option in the Extractor dialog window allows you to *Join Nets With Same Name.* This can be useful when running extractions on portions of a large design, where two wires will be joined by some as-yet undrawn connections. ***Beware*** that this should **not** be used on final designs, as this might hide real errors.

4.4 Electrical Rules Check (ERC)

ERC is another verification step which attempts to determine that no unusual features exist in the extracted view of a circuit layout. This procedure tries to locate floating devices, devices connected to only one net, floating nets short-circuits, and one-terminal nets.

1. While editing an extracted cell view, select **Verify→ERC** and an *ERC* popup dialog window appears.

2. Check the cell information on the form, and ensure that the library, cell and view names correspond to the extracted cell view you wish to check. If the fields are blank, you can click on **Sel by Cursor**, then click in the window displaying the extracted cell view.

3. If the *Rules File* is blank, type **divaERC.rul** in that space.

4. If the *Rules Library* is blank, click the check-box beside the blank field and type **cmosp35** in the field.

5. When ready, click on **Run**. You will be prompted when the ERC check is done.

To check what rules are violated:

1. Select ***Display Errors*** from the ERC form to highlight the ERC errors; select *Explain Error* and click on the marked error.

2. The object name, type and the ERC error message associated with the selected error will be displayed on the CIW or a pop-up window of your choice.

3. Make necessary corrections on the layout cell view, run DRC, re-extract and run ERC again.

4.5 Layout Versus Schematic (LVS) Verification

Cadence provides additional verification tools that require additional designer input in order to verify a layout, or a schematic. Since typically a layout is drawn based on a circuit schematic, Cadence has a tool which is capable of comparing an extracted layout versus the original schematic.

1. Display the extracted and schematic views of a cell side by side (in two windows)

2. In the schematic or extracted view, select **Verify→LVS**, to bring up the Artist *LVS* popup dialog window.

3. Change the default *Run Directory* to one you desire, otherwise leave the rest of the form intact unless the schematic or extracted cellview names are not what you wanted to compare.

4. If the Rules File entry is blank, set it to **divaLVS.rul**.

5. Check all entries and click **Run**. You will be prompted when LVS is done.

6. While LVS is running, you can check the progress by viewing the log file:

   a. Click the *Info* button at the bottom of the Artist *LVS* dialog window, a new *Display Run Information* popup dialog window will appear.
   b. Click the *Log File* button to monitor the LVS progress.

7. After the LVS is done, click the *Output* button in the *Display Run Information* or the Artist *LVS* dialog window. The output log file of the LVS run will be shown in a text window. Near the bottom of the file you will find an indication as to whether the two circuit representations match or not.

8. To display unmatched nets, instances, parameters, etc., click the *Error Display* button in the Artist *LVS* dialog window.

9. Alternatively, you can use cross-probing to look at which devices, nodes, terminals, etc., in the extracted view correspond to the one you've selected in the schematic view, or vice versa:

   a. Select **Verify→Probe** in the extracted or schematic view.
   b. Set the *Probing Method* to cross probe matched.
   c. Click anywhere in the schematic or extracted view to set it as the current window.
   d. Click the *Add Device* or *Add Net* button and select the device or net you want to look at in the schematic.
   e. The corresponding device or net will appear highlighted in the extracted view, if they have been found to match during LVS.

   Note: If you are using cells from the CMC-supplied black-box libraries, you must use macro LVS because there are no schematics for the black-box cells. Please refer to documentation in the files /CMC/kits/cmosp35/doc/CMOSP35lvsFlow or /CMC/kits/cmosp18/doc/CMOSP18lvsFlow for details.

4.6 Export and Import STREAM Data

STREAM (also known as GDS, GDS2, or, more properly, GDSII) format is an industry-standard language for describing mask layouts. This is a binary-data format (i.e. not generally readable as text), and as such can represent a complex design in a very compact form. Designs which are going to be submitted for fabrication through CMC or commercial foundries must be exported from Cadence in this format before submission.

> **Note:** During translation between Cadence data format and STREAM format, a file called the *layerMap table* is used to indicate how layers are translated between the two systems. The *layerMap* file generally translates only the "drawing" layers, and not the "pin" or other layers, so during translation you might see warnings about ignored "layer-purpose" pairs. In most cases these are safe to ignore, but if you have any questions, please contact VRG staff for assistance.

4.6.1 Generating STREAM data (STREAM-out)

1. In the CIW, select **File→Export→Stream**; the *Stream Out* popup dialog window appears.

2. Enter the following values:

Field name	**Value**
Library Name	TestLib (or your library name)
Top Cell Name	inverter (or your cell name)
View Name	layout
Output	StreamDB
Output File	inverter.strm
Units	micron
Error Message File:	PIPO.LOG.inverter.streamout

3. Click on the *User-Defined Data* button near the top of the *Stream Out* window and enter these values into the *Stream Out User-Defined Data* popup dialog window:

Field Name	**Value**
Convert Pin To	geometry
Layer Map Table	/CMC/kits/cmosp35/cmosp35.strmMapTable

   Click **OK**.

4. Click on the *Options* button in the *Stream Out* dialog window. Examine the various options presented; default values should work fine for now; click **OK**.

5. Click on **OK** in the *Stream Out* popup dialog window to begin the translation process.

6. Upon completion, a pop-up window will appear indicating success or failure of the conversion, and a brief summary will appear in the CIW.

   Note: Always examine the contents of the Error Message File specified in the Stream Out popup dialog window for warning and error messages.

4.6.2 Importing STREAM data (STREAM-in)

There are some apparent inconsistencies in the way the STREAM-In procedures handle libraries which are created using the *Attach to existing techfile* method (see Section 2.9). The actual cause of this problem is that the STREAM-In procedure tries to open the technology file for the library to which the target library is attached, in **append** mode (i.e. STREAM-In tries to modify the system technology library, which could be dangerous in a multi-user environment). Because of this problem, we recommended that a library used as the target for the STREAM-In procedure be created using the *Compile a new techfile* method. After the STREAM-In procedures is complete, we can attach to an existing technology file.

1. Create a STREAM-In target library: In the CIW, select **File→New→Library**

2. In the pop-up fill form, type the name of the library to be created, **streamin**, and select the *Compile a new techfile* option. Click **OK** to continue.

3. In the *Load Technology File* popup dialog that appears, enter the full file name of the technology file to be compiled: usually in the form /CMC/kits/cmosp35/cmosp35.tf

4. Click **OK** to continue.

5. Examine the CIW for any error messages during the creation of the new library.

To import STREAM data into this library:

1. In the CIW, select **File→Import→Stream**; the *Stream In* popup dialog window appears.

2. Enter the following values:

Field Name	**Value**
Input File	inverter.strm
Top Cell Name	inverter (or, leave this value empty)
Output	Opus DB(i.e. Cadence database format)
Library Name	streaminTest
ASCII Technology File Name	/CMC/kits/cmosp35/cmosp35.tf
Scale UU/DBU	0.001
Units	micron
Error Message File	PIPO.LOG.inverter.strmin

3. Click on the *User-Defined Data* button near the top dialog window and enter the following value into the *Stream In User-Defined Data* dialog window:

Field Name	**Value**
Layer Map Table	/CMC/kits/cmosp35/cmosp35.strmMapTable

   Click **OK**.

4. Click on the Options button near the top of the Stream In dialog window and examine the various options; Click OK to confirm the default options.

5. Click on OK on the Stream In dialog window to begin the translation process.

6. Upon completion, a pop-up window will appear indicating success or failure of the conversion, and a brief summary will appear in the CIW.

   Note: Always examine the contents of the Error Message File specified in the Stream Out popup dialog window for warning and error messages.

7. Open the *Library Manager* using the CIW window, using **Tools→Library Manager**. In the *Library Manager* popup dialog window, select **View→Refresh** to ensure that your Cadence session has re-read the information on disk, which was created during the STREAM-in operation. Browse the streaminTest library and examine the top cell, inverter, to ensure that all mask features were transferred properly.

4.7 Export and Import CalTech Intermediate Form (CIF)

CIF was used extensively in the early days of CAD software development at universities in fields related to VLSI design. Rather than being fully integrated into a complete design framework and sharing a common database (like modern design tools), early university-developed tools were stand-alone entities. Each of the components represented now in Cadence DFII (e.g. layout, DRC, Extraction) were individual computer programs designed to read mask layout information in the form of CIF, process the information, then output more CIF data or SPICE netlists or other information. The main advantage of CIF over industry-standard description languages is that it is readable by humans, and thus easy to parse by humans (with some basic knowledge) and early computer software. Some modern software still is not capable of reading or writing STREAM data but can read or write CIF, but these packages are becoming quite rare.

We mention CIF here, only to indicate that Cadence DFII can import and export data in CIF format. See on-line documentation for details if necessary, or contact VRG staff.

4.8 Verification using Cadence Assura or Mentor Calibre

For deep submicron technologies, the Cadence "diva" verification tools are less efficient than newer verification tools, notably Cadence Assura and Mentor Calibre. Design-kits for these newer technologies will include documentation describing the usage of these tools.

Chapter 5 Cadence: Analog Simulation

Cadence provides several interfaces to simulators supported by Cadence and other vendors. In this chapter we examine the Analog Environment (also known as "Analog Artist") for simulation of schematic designs and extracted layouts using Cadence's SPECTRE and Synopsys's HSPICE simulators.

5.1 Creating a Test Bench Schematic

A test bench is the analog to a real physical laboratory. In a real lab, there are power supplies, function generators, and oscilloscopes; in our virtual test bench we use similar instruments, represented by various icons, to stimulate and observe the design under test (DUT). Typically, a test bench is used as a starting point for a schematic or extracted layout simulation.

Before you can run a Spice or Spectre simulation, you will need to create a symbol of the design to be simulated, the Device Under Test (DUT). This symbol for the DUT will then be instantiated in the test bench schematic.

1. Create a symbol for the *inverter* schematic, if one does not exist already. Open the *inverter schematic* and select **Design→Create Cellview→From Cellview** to create a symbol, as described in Section 3.3.

2. Create your test bench schematic:

   a. Create a new schematic cell view in your *TestLib* library, give it a unique name to remind you that it is a test bench schematic for the inverter schematic: for example **tb_inverter**.
   b. Place an instance of the **inverter** *symbol* into the test bench schematic.
   c. Place an instance of a capacitor; you can find analog components in the *analogLib* library. Place an instance of the **cap** symbol, assign the capacitor a variable capacitance, ***cload***.
   d. Connect the capacitor to the output of the inverter.

3. Add signal stimuli and power supplies to the **tb_inverter** schematic. Power supplies and signal stimuli can often be found in the library named after the technology; if they don't exist there, you can find them in the *analogLib* library:

   a. A sinusoid voltage source will be used to stimulate the input of the inverter. Place an instance of the **vsin** symbol in your schematic. Specify the values given in the following table in the *Add Instance* popup dialog window for the *vsin* symbol.

Field	**Value**
AC magnitude	0
AC phase	0
DC voltage	0
Offset voltage	1.65 V
Amplitude	1.65 V
Frequency	100K

   b. The inverter logic gate will not function without a DC power supply. Place an instance of the **vdc** symbol in your schematic. Specify the *DC voltage* to be a variable, **vdc**.
   c. Spice and Spectre simulations require that a reference node, also known as a ground node, be explicitly specified in the test bench. Place an instance of the **gnd** symbol in your schematic or create a net with the name **gnd!** -- the effect is the same.

Note: If you used global signal names for the power supplies (e.g. *vdd!* and *vss!*) then you will need to name the nets connected to the DC power supply and ground *vdd!* and *vss!*, respectively, in order to establish global power supply net connectivity.

d. Wire up your test bench schematic so it looks like the schematic shown in Figure 5.1:

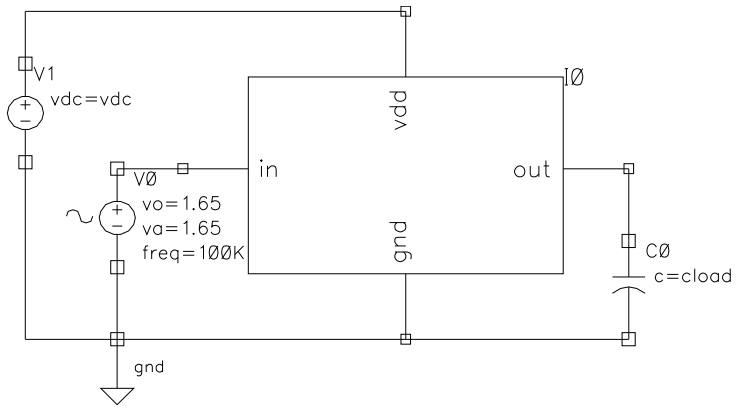


Figure 5.1: Simple inverter test bench schematic

e. Check and save your test bench.

5.2 Setting up a simulation

Cadence provides a graphical user interface to standard industry circuit simulation tools, such as Spice and Spectre, with which you can specify instance variables before simulation, specify variables you wish to sweep, optimize, etc. during simulation, and display the results of simulation using a GUI plotting environment and even directly on the test bench schematic itself. To open the interface, you can select **Tools→Analog Environment** from the command menu in your test bench schematic.

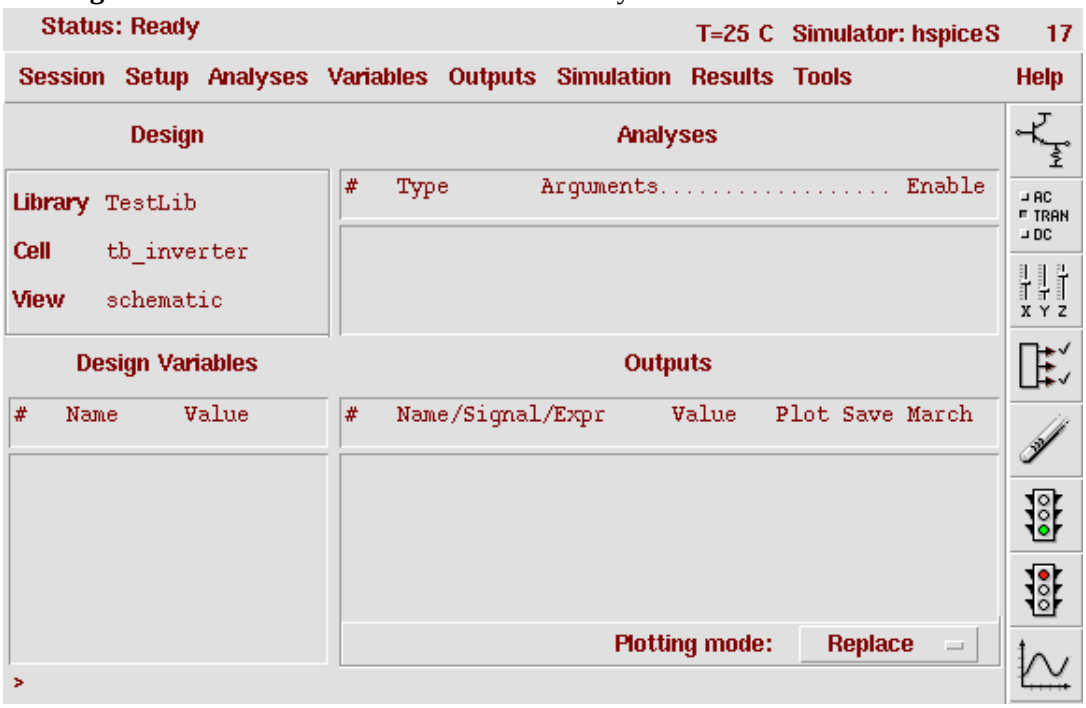


Figure 5.2: Virtuoso Analog Environment GUI

The window shown in Figure 5.2: consists of four quadrants. The top-left quadrant specifies the schematic the window is tied to; each window session can only be tied to one schematic.

The bottom-left quadrant allows the designer to specify values for any Design Variables, that is any variables specified while instantiating design components.

The top-right quadrant allows the designer to specify what type of circuit analysis to be run during simulation: DC analysis, AC analysis, transient analysis or noise analysis. You can specify several of the same types of analyses and then toggle whether they are enabled.

The bottom-right quadrant allows the designer to specify types of measurements to be captured during simulation; voltage information for all nodes is captured automatically during simulation. You can specify what types of measurements should also be plotted or displayed on the schematic automatically after the simulation completes.

The Analog Environment is well integrated within the Cadence design environment; if asked to specify instances, power supplies, or nodes by any of the popup dialog windows, you can simply point and click on the item in your schematic directly. You can also traverse the design hierarchy, starting at the test bench schematic to select nodes or items within your design hierarchy.

5.2.1 Including technology-specific device models

Technology specific device models are not automatically linked by the Analog Environment; you must specify them manually when you begin a new Analog Environment session. Hence, it's a good idea to save your session, so you won't have to redo these steps.

> **Note:** Each Analog Environment session is specific to only one schematic. Hence, you will not be able to load session settings from another schematic.

1. Select **Setup→Simulator/Directory/Host**, which will bring up the *Choosing Simulator/Directory/Host* popup dialog window.

2. Select the target simulation engine from the *Simulator* drop down list: *hspiceS* or *spectreS* is covered by this tutorial.

   > **Note:** Newer design kits use *hspice* and *spectre*, rather than *hspiceS* and *spectreS*, the so-called "socket interface" to simulators; the newer interfaces provide better integration into the Cadence tools.

3. The *Project Directory* field allows you to specify where all simulation files are to reside. If your design is large, then it will be beneficial to use `/tmp` as the root of your project directory; `/tmp` is your host's local temporary storage.

   > **Note:** Using non-local network-mounted directories will severely limit the speed of your simulation for large designs. Remember that, by default, voltage values for every node in your netlist are stored during simulation; this can generate quite a lot of data during simulation!

4. Select **Setup→Environment**, which will display the *Environment Options* popup dialog window. Fill in the following values depending on which simulator and technology you are using.

hspiceS: Set *Include/Stimulus File Syntax* to **hspice** and specify the *Include File* as:

CMOSP35: /CMC/kits/cmosp35/models/hspice/icdhspice.init
CMOSP18: /CMC/kits/cmosp18/models/hspice/icfhspice.init

spectre: Set *Include/Stimulus File Syntax* to **spectre** and specify the *Include File* as:

CMOSP35: /CMC/kits/cmosp35/models/B3V/icdspectre.init
CMOSP18: /CMC/kits/cmosp18/models/spectre/icfspectre.init

5.3 Simulating a Design

5.3.1 Initializing Design Variables

If you know what simulation variables need to be defined you can enter their values by selecting **Variables→Edit**, which will bring up the *Editing Design Variables* popup dialog window. You specify variables by entering a variable name into the *Name* field, a corresponding value into the *Value (Expr)* field, and clicking **Add**. Specify the following variable values:

Name	Value
cload	1p
vdc	3.3

Alternatively, you can use **Variables→Copy From Cellview** to "import" a list of variable names from the cellview.

5.3.2 Choosing a simulation analysis

Transient, AC and DC analysis can be run simultaneously. Units for analysis are in seconds, Volts, Amps, and Hertz depending on the analysis type.

1. Select **Analyses→Choose**, bringing up the *Choosing Analyses* popup dialog window.

2. Select between transient, AC and DC analyses by toggling their radio buttons.

3. Specify three analyses: a transient analysis, an AC analysis, and a DC analysis with the following parameters:

Transient	From **0** to **20u** by **0.5u**
AC	From **1k** to **1G**, **logarithmic**, **10** points per decade
DC	From **0** to **6** by **0.2**

4. For a DC analysis, click on the **Select Source** button and choose the DC power supply in your test bench schematic. The *Source Name* field should be filled in automatically with the DC power supply's instance name.

5.3.3 Plotting Outputs

Voltage simulation data will be saved automatically for each node in the simulation. We will now specify which nodes we would like to have plotted when the simulation completes.

1. Select **Outputs→To Be Plotted→Select On Schematic**, which should bring your test bench schematic into focus.

2. Zoom out to fit the schematic to screen, shortcut key **f**.

3. Click on the input and output **wires** in the schematic, they will change color to indicate they have been selected. You have just chosen to plot the input and output voltage of the inverter.

4. To plot currents, click on the **terminal** you are interested in measuring the current going into or coming out of. A circle should appear around the terminal to indicate that it has been selected.

   Note: Double clicking on an instance will automatically select all terminals of the instance.

5. Press **ESC** when you have completed selecting measurements to plot.

5.3.4 Saving Outputs

Currents, and some other outputs besides voltages, are not by default automatically saved during simulation. In order to be able to plot or view these outputs, you need to explicitly specify that they are to be saved during simulation.

> **Note:** You can disable the automatic saving of all node voltages by selecting **Outputs→Save All** and toggling the *Select all node voltages* check box in the *Keep Options* popup dialog window that appears.

1. If you have already specified measurements to be plotted, simply double click on any output in the *Output* list that do not have a *yes* or *allv* (all voltages) in the *Save* column. A *Setting Outputs* popup dialog window will appear.

2. Make sure the **Saved** radio button is checked in the *Will Be* section of the dialog window.

3. Click **Change** to save the modification to the output.

You can also select outputs you wish to save by selecting **Outputs→To Be Saved→Select On Schematic** and selecting the outputs directly on the schematic itself.

> **Note:** It does not matter if you specify the voltage outputs to be saved or not in this list. Notice the *Save* column for voltage nodes never changes from *allv.*

5.3.5 Selecting marching outputs

Sometimes, for long simulations, it is convenient to monitor how outputs are developing during the simulation, allowing for early termination of a simulation that is not proceeding as planned. This can be accomplished by selecting some outputs to be *Marching Outputs*:

1. If you have already specified measurements to be plotted, simply double click on any output in the *Output* list that do not have a *yes* or *allv* in the *March* column. A *Setting Outputs* popup dialog window will appear.

2. Make sure the **Marched** radio button is checked in the *Will Be* section of the dialog window.

3. Click **Change** to save the modification to the output.

You can also select outputs you wish to march by selecting **Outputs→To Be Marched→Select On Schematic** and selecting the outputs directly on the schematic itself.

> **Note:** You will not see marching outputs if your simulation does not take longer than approximately five seconds to run.

5.3.6 Running a simulation

Once you have chosen the type of simulation you'd like to run, specified any outputs you'd like to plot, save or march, and specified any design variables you can proceed to run a simulation. Running a simulation is easy, simply select **Simulation→Run** or click on the **green light**.

Status messages are reported both in the simulation and the CIW windows. Marching outputs will appear in a separate window after the simulation has started. You may hear an audible tone once the simulation has completed, and you should see a status message in the simulation window indicating that the simulation was a success. Any outputs you have chosen to be plotted will be automatically shown once the simulation has completed.

5.3.7 Unsuccessful simulations

Occasionally you might see messages indicating that a simulation failed. One of the most common causes for a failure is that one or more of your MOS devices could not be matched with an appropriate Spice model. Spice *.MODEL* syntax provides a mechanism for specifying that a model is valid for a range of gate length and width; if you see messages indicating *unknown use of pch* or *nch* message, then it probably means that one of the devices has a length and width combination with no corresponding model. Verify that all of your devices fall within the ranges specified by your technology node.

5.3.8 Viewing the netlist

It is not necessary to generate or view a netlist before running a simulation as it will be generated automatically when required. However, we will create and view a netlist as an exercise; you might need to generate a netlist manually to help in debugging simulation runs, or during research projects when you wish to generate a netlist from within Cadence, but use Spice in stand-alone mode.

Lets look at a raw or informational netlist:

1. Select **Simulation→Netlist→Create Raw** in the Virtuoso *Analog Environment* dialog window.

2. Overlapping text windows will display the raw netlist information. There is netlist for the global design, and one for each level of the design hierarchy. Examine these files.

3. When done, close all netlist windows.

Let's take a look at the final simulation netlist,

1. Select **Simulation→Netlist→Create Final** in the Virtuoso *Analog Environment* dialog window.

2. A window displays the final netlist information. Observe the hierarchical structure and sub-circuit calls.

3. When done, close the netlist window.

Warning and error messages during netlisting

During netlisting, be sure to watch the CIW any warning or error messages, which should be examined carefully, and corrected. Common warning messages that may be safely ignored are ``UNABLE TO OPEN FILE: pch.m'' or ``UNABLE TO OPEN FILE: nch.m''. Some design kits provide separate model files (*.m) containing Spice or Spectre model information, but most of the design kits available under /CMC/ kits have all model information in one file, so they do not use this format.

Common warning messages that need to be examined carefully include: ``reference 0:pch not found'', which indicates that the automatic model selection (based on the width and length specified for one of the devices in the design) has failed. To correct this problem, locate the device indicated (in this case, a p-channel MOSFET, with instance-ID 0), examine its properties and ensure that the length/width combination corresponds to one of the models specified by your technology's model files.

5.4 Displaying Output Waveforms

After a successful simulation, you can always add more voltage nodes to the list of outputs to be plotted. You can do so by selecting **Outputs→To Be Plotted→Select On Schematic** and then selecting nodes you'd like to plot directly on the schematic.

You can choose wires that are in a lower level of your design hierarchy, as well:

1. Simply **click-and-hold** the **middle mouse button** on the instance you'd like to descend into; select **Descend Read...** from the popup menu that appears, which will bring up the *Descend* popup dialog window.

   Note: You can also hover the mouse cursor over the instance you wish to descend into and use the **e** keyboard shortcut.

2. Select **Schematic** as the *View Name* from the drop down list in the *Descend* dialog window and click the **OK** button.

3. Select nets in this cellview, or keep descending the design hierarchy if you require.

4. You can return to the previous level in the design hierarchy by **pressing-and-holding** the **middle mouse button** in the schematic window and selecting **Return** or use the keyboard shortcut **CTRL+e**.

To plot the chosen outputs, select **Results→Plot Outputs** then **DC**, **Transient**, or **AC** depending on which simulation analysis results you'd like to view.

5.4.1 Printing Simulation Waveforms

In the waveform window, select **File→Print**, which will bring up the *Print* dialog window. Choose an appropriate printer or check off the **Print To File** check-box.

In the *Page Setup* tab, select *Media Size* to be **Letter**, and your preferred orientation. In the *Annotations* tab you can select what extra information you would like to have printed along with your plot.

5.5 Waveform Calculator

You can use the waveform calculator to build and display expressions containing your simulation output data; enter expression that can contain node voltages, port currents, model parameters, etc. The calculator can function either in *reverse Polish notation (RPN)* or using *infix notation*.

To start the calculator select **Tools→Calculator** in the Virtuoso *Analog Environment* dialog window. You can toggle between RPN and infix notation by selecting **Options→Set RPN** in the *Calculator* window.

Transient, AC, or DC voltage waveforms can be added to the expression field by selecting the appropriate tab (*tran*, *ac*, or *dc*); click on one of the radio buttons that appear in the tabbed list (e.g. *vt* or *it*, denoting voltage and current transients respectively); then, select a net or node from the schematic window that is brought into focus.

To plot an expression, simply click the button that looks like a squiggly line on graph paper, located under the expression field and to the left of the function window.

You can also save an expression to memory, for later recall; simply select **Memories→Table→New Memories**. This will add the current expression to the expression memory table, where you can also give a unique mnemonic for the expression. You can edit the equations stored in memory by selecting **Memories→Table→Edit**. To add a stored equation into the expression field, select **Memories→Select** and choose the mnemonic for the equation you wish to add. You can also save to disk and load from disk your saved equations by using **Memories→Load** and **Memories→Save**, respectively.

5.6 Parametric Analysis

The Virtuoso Analog Environment features a very useful tool called *parametric analysis*, which lets you sweep design variables during a simulation. You can display the simulation results in a family of curves. To perform a transient parametric analysis:

1. Select **Analyses→Choose**, and select a transient analysis from **0** to **10u** by an increment of **100n**. Disable all other analyses.

2. To limit the amount of simulation data that will be saved, disable the automatic saving of all node voltages. Select **Outputs→Save All** and uncheck the *Select all node voltages* check-box in the *Keep Options* popup dialog window.

3. Enable the saving of the in and out signals, by selecting **Outputs→Setup** and modifying the properties of the in and out signals appropriately.

4. Select **Tools→Parametric Analysis**, which will bring up the *Parametric Analysis* dialog window. The following steps will relate to this window.

5. Select **Setup→Pick Name for Variable→Sweep 1**, which will bring up a selection window with a list of design variables for your test bench.

6. Double click on **cload** to select it as the variable to be swept.

7. Enter the following values as sweep parameters for the *cload* variable:

Field	Value
From	1p
To	1n
Total Steps	5

8. You can see a list of design variables that will be swept, along with the values they will take during the simulation by selecting **Analysis→Show Sweep Sets→All**; click **OK** when you've done examining the list.

9. Optionally, you can add another range to sweep over:
   a. Change the *Add Specification* cyclic field to **range**. Once you do so it will return back to *Add Specification*, but a second range entry will appear.
   b. You can use these steps to add additional disjoint parametric ranges to be swept by the chosen design variable.
   c. Select **Setup→Delete Range Specification** and double click on the added range, in the window that pops up, to delete it.

10. To run the parametric analysis, select Analysis Start. Status messages will appear in the *Parametric Analysis Window* as well as in the Virtuoso *Analog Environment* and CIW windows.

11. Explore the results in the waveform window; close the waveform window and the *Parametric Analysis* window when you're done.

5.7 Post-Layout Simulation

To simulate a physical layout, you must first run layout extraction to extract the devices, connectivity and layout parasitics. Before extraction, make sure you have all the input, output, power and ground pins on the proper pin layers at the top hierarchical level.

The steps involved in post-layout simulation are similar to those in the schematic simulation described earlier. In fact, the same testbench schematic can be used in both cases. However if you do not have a schematic for the layout, you can still create a symbol from the extracted cellview and place it in a testbench schematic as described earlier.

To simulate an extracted layout using your schematic test bench,

1. Open your test bench schematic, and start the Virtuoso Analog Environment.

2. Select **Setup→Environmen**t in the Virtuoso Analog Environment window.

3. The *Switch View List* specifies the order of views Cadence searches in order to generate net lists. Add the word **extracted** before **schematic**; this will generate netlists of sub-circuits using the extracted view, if available, and the schematic view, if the extracted view cannot be found.

4. The rest of the simulation proceeds as normal.

5.7.1 Generating a symbol from extracted view

To simulate a design for which you have no schematic, you can generate a symbol from the extracted cell-view:

1. Open the extracted cell view.

2. Select **CMC Skill→Layout/Extract→Generate pin-only schematic**, which will create a schematic consisting of only pins defined in the layout. This pin information is all that is required to generate a symbol.

3. Open the generated pin-only schematic and create a symbol cell view by selecting **Design→Create Cellview→From Cellview**.

4. Create a test bench schematic, and include an instance of the generated symbol. The rest of the simulation proceeds as explained previously.

Chapter 6 Digital Simulation

In previous chapters we discussed tools used mainly for the design of analog circuits. Here we change focus to digital logic design, and introduce tools from Cadence that are available for simulation at higher levels of abstraction, where systems are described in Hardware Description Languages (HDLs) like Verilog and VHDL. These HDL descriptions may be developed for various levels of abstraction, such as algorithmic, register-transfer level (RTL), gate-level, or switch-level.

Before simulating designs that are described using HDLs, the source files must be *compiled*, to perform semantic and syntactic checking and, if error-free, to generate an internal representation for each module described in the source files. After source files are compiled, *elaboration* builds a design hierarchy based on the instantiation and configuration information in the design, establishes signal connectivity, and computes initial values for all objects in the design. The elaborated design may be simulated, and other tools may be used to help analyze or debug problems with the original source files. Each of these tools may be run independently of the others, but Cadence supplies a graphical user interface called **NClaunch** to help organize your design files.

6.1 Environment settings

In order to be able to access the Cadence NC tools, you must

```
source /CMC/tools/CSHRCs/Cadence.IUS
```

6.2 Starting NClaunch

Create a new work directory for your digital simulations, change directories to the new location, then copy a sample VHDL file there:

```
% mkdir DigitalSim
% cd    DigitalSim
% cp    /CMC/kits/VRGlocal/demo/vhdl/converter.vhd  converter.vhd
```

The first time you run NClaunch in a work directory, invoke it using

```
% nclaunch -new &
```

A dialog window should appear, asking you to select a run mode. For Verilog-only designs, you may select *Single Step*; for all other designs -- either VHDL-only or mixed VHDL/Verilog designs -- select *Multiple Step.* (Examples used in this manual will use VHDL, so click **Multiple Step**.)

The *Open Design Directory* dialog window appears. Click on **Create cds.lib File...** to create a library mapping file in the current directory. In the *Create a cds.lib File* dialog window, click **Save**, then in the next *New cds.lib File* dialog window, click **OK**. Back in the *Open Design Directory* dialog window, click **OK** to complete the initialization.

The main NClaunch window appears. The left pane shows the contents of the current working directory in a *File Browser.* The right pane shows a *Library Browser.* The *I/O* pane at the bottom will show warnings, errors, and other status messages, and allows you to type commands to be executed.

6.3 Compiling Source Files

This source file `converter.vhd` represents a simple decoder for an LCD seven-segment display, as shown in Fig. 6.1.

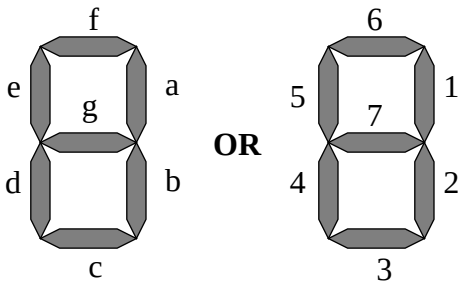


Figure 6.1: LCD Seven-segment display

This source file must be compiled to generate an internal representation usable by the Cadence tools. To do this, you can double-click on each VHDL file's name in the File Browser or click on the **VHDL** icon in the tool bar near the top of the NClaunch window. If you need to set special compile options, left-click the source file and use **Tools→VHDL Compiler...** Try double-clicking on **converter.vhd** and examine the status messages in the I/O pane at the bottom. You should see a few lines indicating to you a version number, how much memory was used in various phases of the compile, and how much CPU time was used. In this case, you will also see messages that indicate that there are some syntax errors in the source file, so use a text editor (or click the icon beside the *Browsers* banner near the top of the **nclaunch** window to open an edit session) to correct the errors. When done, save the file and compile again. (You need not close or restart **nclaunch**.) Repeat repairs until there are no error messages. Upon successful completion of this step, the **worklib** folder under the *Library Browser* will hold a copy of the compiled source.

Now copy a VHDL testbench to your directory:

```
% cp /CMC/kits/VRGlocal/demo/vhdl/tb_converter.vhd  tb_converter.vhd
```

This testbench stimulates the `converter` module, providing a clock signal and a "walking ones" pattern to the inputs of the *converter*. If this file does not appear in the File Browser pane, you can click the *Refresh* icon beside the *Browsers* banner near the top left of the **nclaunch** window. Compile this testbench source file by double-clicking on its name.

> **Note:** You can use **ncvhdl** outside of the NClaunch system to compile source files in VHDL format, or **ncverilog** to compile source files in Verilog format.to correct the errors

6.4 Elaborate the Design

The next step is to elaborate the design, to build the hierarchy as specified in the testbench file. In the right pane, the Library Manager shows the **worklib**, which holds the compiled modules. Click the **+** sign beside it to view its contents. Select the configuration for the top level of the design, **cfg_tb_converter** and click on the **Elaborate** icon in the tool bar near the top of the NClaunch window, or use **Tools→Elaborator...** to modify options. If error-free, the Elaborator saves a copy in the **Snapshots** folder. Click on the **+** sign beside **Snapshots** to view its contents.

> **Note:** You can use **ncelab** outside of the NClaunch system to elaborate a design.

6.5 Performing a Simulation

A design that has been elaborated can be simulated. Select the elaborated design from the **Snapshots** folder, and click on the **Simulate** icon near the top of the NClaunch window. This will open another graphical user interface, this time to **SimVision**, the simulator. Two windows will open: a *Console* window, and a *Design Browser* window. The left pane of the *Design Browser* shows **simulator** as the top level of the design hierarchy, with the elaborated design ("**:**") directly beneath it. Left-click on the **+** beside your elaborated design to examine the hierarchy of the design. SimVision allows you to probe signals within the design, so you can examine logic levels at various points in your design.

Select **UUT** in the Design Browser. The right pane shows the signals available, and indicates the type (input, output, inout, etc.) of each signal. In the upper right corner of this window, click on the icon indicating waveforms (white lines on a black background) to open a waveform viewer for the selected signals displayed in the right pane. In the Console Window, type **run 300ns** to run the simulation for a fixed period of time. (The testbench will actually run forever if you click the **run** button or use **Simulation→Run**.) Change the zoom level of the waveform window by typing **ALT-=** or use **View→Zoom→Full X**. Verify that the converter is operating correctly.

> **Hint:** You can re-arrange the order of waveforms with your mouse, using middle-click-and-hold on the signal's name, then drag-and-drop up or down to the desired location.

Familiarize yourself with the other possible operations available with this tool, by exploring the banner menu items and examining the documentation.

> **Note:** You can use **ncsim** to start the simulator outside of the NClaunch system.

6.6 Associated Cadence Tools

A full list of the tools available under the NClaunch system is available within the Cadence on-line documentation, accessible using **cdsdoc**.

6.7 Synopsys Simulation Tools

Tools for digital logic simulation are also available from Synopsys. One set of tools, **VCS-MX**, is accessible using this "source" command:

```
% source /CMC/tools/CSHRCs/Synopsys.VCS
```

This tool will not be discussed here; documentation is available using the Synopsys On-Line Documentation tool:

```
% /CMC/tools/synopsys/sold/sold &
```

Chapter 7 Logic Synthesis and Optimization

Typically, the sequence of well-defined steps used to synthesize a system's behavioural description in a Hardware Description Language into a gate-level design targeting a specific technology design-kit would be run from a script of commands that have been saved to a text file. In this chapter, we illustrate many of the steps used during synthesis by demonstrating them using a graphical user interface or command-line entry to the Synopsys **Design Compiler**. These steps will be recorded in a file in your work directory, so you may edit this and customize it as required for other projects. The example we will work through here and in the next chapter will use the TSMC **CMOSP18** process technology and the **Artisan 3.0** libraries for this technology, but where possible we will describe the set-up for the TSMC **CMOSP35** technology as well.

7.1 Available libraries

When synthesizing a design from a behavioural (or other high-level) representation of a system, we need to target a specific technology, with libraries of building blocks that have been designed to be fabricated using that technology. The synthesis programs read information from library files and can select appropriate replacements for "generic" building blocks. These libraries can include all of the mask data for all of the cells in the library, but more often when we use commercially designed libraries we see only a "black-box" representation of the library cells, so named because the layout, extracted and schematic views are not available to designers; the details within the cells is hidden from view. This makes it impossible to reverse-engineer, but it also makes it difficult for designers to examine a completed design. The black-box library information, often in the form of an *abstract* view, provides sufficient information about the cells contents (pin locations, metal levels required for connecting to the pins, and blockages or "keep-out" regions indicating areas where over-the-cell routing is prohibited) for the automated synthesis and place-and-route tools to help you prepare designs that are physically correct: the completed designs should violate no design rules, and all timing constraints will be satisfied. In some cases the library information will provide simulatable transistor-level netlists of all library cells, but for advanced technologies this information will be missing.

7.1.1 CMOSP18

The CMOSP18 technology distributed by CMC only contains older black-box libraries for standard cells and the I/O pads. In the examples worked in this chapter we will use a newer set of commercially available black-box libraries known as the **Artisan version 3.0** standard cell and I/O libraries.

7.1.2 CMOSP35

In the CMOSP35 technology distributed by CMC, two sets of standard cell libraries are available: *wcells* and the black-box libraries (*tcb773pwc* and *tpd773pnwc*). The *wcells* library was developed at CMC, while the black-box libraries are commercial libraries. The default technology library path for Synopsys is set to access the *wcells* library. If you wish to use the black-box libraries, you will need to make appropriate changes in the `.synopsys_dc.setup` file mentioned below for this technology.

7.2 Environment Settings

Access to the Synopsys *Design Analyzer*, used for synthesis and optimization, requires that you

```
% source /CMC/tools/CSHRCs/Synopsys
```

The graphical user interface, ***design_analyzer***, and the scripting interface, ***dc_shell***, both require two technology-specific set-up files in your work directory. To set these up, you can copy the appropriate set-up files depending on your technology choice. (Note the space followed by a '**.**' at the end of all of the "**cp**" commands.)

CMOSP18:	**% cp**	**/CMC/kits/VRGlocal/demo/synopsys/.synopsys_dc.setup .**
	% cp	**/CMC/kits/VRGlocal/demo/synopsys/.synopsys_vss.setup .**
CMOSP35:	**% cp**	**/CMC/kits/cmosp35/synopsys/dotfiles/.synopsys_dc.setup .**
	% cp	**/CMC/kits/cmosp35/synopsys/dotfiles/.synopsys_vss.setup .**

In addition, you will need to create a directory named "Work" to hold intermediate files:

```
% mkdir ./Work
```

7.3 Starting *design_analyzer*

1. Invoke the graphical user interface to the synthesis program using

   ```
   % design_analyzer &
   ```

2. Select **Setup→Defaults...** to bring up the *Defaults* dialog window.

   a. Feel free to fill in the *Designer* and *Company* fields.
   b. The *Search Path* specifies a list of filesystem paths that *design_analyzer* will search for any design files required for synthesizing the design. It is filled in using technology-specific filesystem paths indicated in your `.synopsys_dc.setup` file. If you have your design files in other locations, you can add these to this field.
   c. The *Link Library* field specifies the names of libraries that will be linked to your design.
   d. The default values in other fields should be fine. Click **OK** to continue.
   Note: Environmental settings are per session settings, you will need to re-enter them every time you start Design Compiler. To make the respective settings default for all future sessions, edit your `.synopsys_dc.setup` file instead.

3. Select **File→Read...** and select your design files from the drop-down list. A log window echoing load messages will appear. Notice the icon at the center of the *Design Analyzer* window. The expression inside the box indicates that the design has not yet been compiled to a gate level netlist.

4. For a design that consists of multiple files, read in each file, one at a time. Then select the top level module in the *Design Analyzer* window and select **Analysis→Link Design...**

5. After loading all necessary HDL files, it is wise to check the design for errors that might have not become apparent during simulation before synthesis. Select **Analysis→Check Design** and confirm the default check settings. A return value of '1' indicates that the were no warnings or errors in the design.

7.4 Explore the Design Hierarchy

You can, at any time, explore your design hierarchy in *Design Analyzer* by selecting various modules in your design and using the short-cut buttons shown on the left side of the window. From the top, the icons represent

- Designs view: indicates the designs that have been loaded into memory.
- Symbol view: shows a symbolic representation of the design, including input and output ports. *Attributes* (showing design-specific information such as the arrival times and strengths of input signals), and *constraints* (design-specific requirements or goals for the optimizer such as the maximum circuit area or the maximum signal propagation time through the circuit to one or more outputs) may be set while viewing this representation.
- Schematic view: shows the composition of the design. Components may be gates or subdesigns.
- Text view: This is disabled in this version of the *design_analyzer.*

The next two icons allow you to move up or down in the design hierarchy.

7.5 Prepare the Design for Synthesis

Before synthesizing the design, specify any attributes or constraints for your design:

7.5.1 Specify clock pin and signal

If your design includes a clock signal:

1. Select the top level module in your design hierarchy. Display its symbol view.

2. Select the pin corresponding to the clock pin.

3. Select **Attributes→Clocks→Specify**, which will bring up the *Specify Clock* dialog window.

4. Ensure that the *Don't Touch Network* check-box is checked. This will prevent Design Analyzer from optimizing the clock tree by inserting buffers; buffers will be inserted using a different tool, during the floor planning, placement, and routing.

5. Specify the *Period* of the clock, in nanoseconds, and specify the position of the positive *Edge*, also in nanoseconds.

6. Click **Apply** to apply the settings, notice that your clock pin now has a red square wave signal. Click **Cancel** to close the *Specify Clock* dialog window.

7.5.2 Specify operating environment: input drive strength and output loading

Ideal circuits may have their inputs driven strongly, and no output loading. Real world circuitry should be synthesized with more-realistic values. You can specify the input drive strength:

1. Select the top level module in your design, and drop down into its symbol view.

2. Select an input pin, then

- Select **Attributes→Operating Environment→Drive Strength...**
- In the *Drive Strength* dialog window, turn on **Same Rise and Fall** and specify a **Rise Strength** of 30. This value is measured in delay (ns) per load (pF); the drive strength increases as the value approaches 0.
- Click **Apply** and then **Cancel**.

Note: If several pins will have the same drive strength, you can select multiple pins by **SHIFT-left-clicking** on all of the similar pins before specifying the *Drive Strength* and clicking **Apply**.

The load on an output pin will impact the timing of the pin. Therefore, it is a good idea to provide the synthesis tool with an estimated capacitive load each output pin will experience after manufacturing. To specify the load on an output pin:

1. Select the top level module in your design, and drop down into its symbol view.

2. Select each output pin, one at a time, and perform the following

   - Select **Attributes→Operating Environment→Load**, which will bring up the *Load* dialog window.
   - Specify the expected *Capacitive Load* on the pin in **pF**
   - Click **Apply** to apply the capacitive value to the selected output port.
   - With the *Load* dialog window still open, you can select another output pin on your schematic, and specify its output load as well, by specifying the estimated *Capacitive Load* and then clicking **Apply**
   - When completed, click **Cancel** to close the *Load* dialog window.

   Note: If several pins will have the same load, you can select multiple pins by **SHIFT-left-clicking** on all of the similar pins before specifying the *Load* and clicking **Apply**.

7.5.3 Specify area constraints

1. Select the top level module in your design hierarchy.

2. Select **Attributes→Optimization Constraints→Design Constraints**. The *Design Constraints* dialog window will appear.

3. Set the *Max Area* field to **0**. This will force Design Compiler to optimize the design for the smallest silicon area.

4. Click **Apply** to confirm the settings, and then **Cancel** to close the dialog window.

   Note: On some versions of *design_analyzer*, this command fails with an error related to fault coverage, due to a problem with one of the installed scripts. To avoid this error, you can type this command instead, into the bottom line of the *Command Window*:

   ```
   set_max_area  0
   ```

7.5.4 Specify timing constraints

1. Select the top level module in your design hierarchy.

2. Double-click on your top-level module to descend into its symbolic or schematic view.

3. Select **Attributes→Optimization Constraints→Timing Constraints**. The *Timing Constraints* dialog window will appear.

4. Make sure the check box for *Same Rise and Fall* is selected.

5. Specify the *Maximum Delay* to be **0.** This will force Design Compiler to optimize the design for the fastest possible path delay, and hence clock frequency.

6. Click **Apply** to confirm the settings, and then **Cancel** to close the dialog window.

7.5.5 Conflicting optimization goals

We have just specified potentially conflicting optimization goals: a minimum size circuit with a maximum clock frequency. By default, Design Compiler tries to satisfy these goals by optimizing for timing, then design area. If you'd like to change the default optimization goal priorities, refer to the on-line documentation.

7.5.6 Multiple use of sub blocks

If you call a sub-block more than once in the top level of a design, the Design Compiler will ask you to execute the **uniquify** command (**Edit→Uniquify→Hierarchy**) before synthesizing the design. If you run this command, Design Compiler will generate independent blocks, each with a unique suffix. Each block will then be optimized individually. For example, if you use a full adder block, *FA*, several times in a multiplier design, the uniquify command will generate a set of individual full adders with names *FA1*, *FA2*, etc.

7.6 Synthesize your design

1. Select **Tools→Design Optimization...** which will bring up the *Design Optimization* dialog window.

2. Select the desired *Map Effort*. A high mapping effort will specify that *Design Compiler* should spend more time trying to meet the specified design optimization goals.

3. Do not select *Verify Design*. In theory, this option performs automatic RTL versus gate level design comparison. However, it takes an extremely long time, especially at higher *Verify Effort* settings.

4. Click **OK** to begin the optimization. As the optimization progresses, the Command Window will be updated with the synthesis progress.

When selected, *Boundary Optimization* allows Design Compiler to make logic and gate level optimizations across module boundaries. If this option is not checked, each module will be optimized irrespective of what it is connected to, or what modules it contains inside. However, boundary optimizations might change the pin-outs of some modules. Refer to the on-line documentation before using *Boundary Optimization*, especially when using external black-box modules, such as memories or black-box standard-cell libraries.

7.6.1 Evaluate optimized results

1. Select the top-level module.

2. Select **Analysis→Report**, which will bring up the *Report* dialog window.

3. Select the check-boxes corresponding to the information you would like the report to provide, such as *Constraints, Area* and *Timing*.

4. Click **Apply** to have *Design Analyzer* generate a report.

The report will provide you with information such as what the design constraints where, and whether they have been met. Other information, such as the longest path (critical path) in your design is provided as well. To see the critical path in your design, bring up the schematic view of your design and select **Analysis→Highlight→Critical Path**.

At this point, you can save your design as **converter_gates.v** or **converter_gates.vhd**

A script is available in `/CMC/kits/VRGlocal/demo/synopsys/converter_build.script` which contains directives which, when run using **Setup→Execute Script...** will build these two files.

7.7 Testability

At this stage of the design, we are ready to add some features to the design which will be useful after our chip has been fabricated, packaged, and returned to us for testing. At this time, these features are beyond the scope of this manual.

If you wish to investigate these for your own projects, some *Design For Test* Tools are available here from Mentor. To access these, you may

```
% source /CMC/tools/CSHRCs/dft
```

Some of the tools available include

- **dftadvisor***:* for *Scan Chain Insertion:*
- **fastscan***:* for *Automatic Test Pattern Generation*

Some sample files for this design are available in `/CMC/kits/VRGlocal/demo/dft/`

7.8 Adding Input and Output Pads

The digital logic on your microchip will inevitably have to communicate with the outside world. This communication is facilitated via buffers and pads, which could include high current inverters with over-voltage or electrostatic discharge diode (ESD) protection. Information about the types of pads provided for the *CMOSP18* technology is provided in the file

```
/CMC/kits/artisan.3.0/FE/doc/
```

Information about the types of pads provided for the *CMOSP35* technology is provided in

```
/CMC/kits/cmosp35/doc/blackbox/tpz773pn_150a/tpz773pntc/html/index.html
```

Pads can be manually inserted into your HDL code before synthesis, by using the appropriate model name for the pad. Alternatively, if the libraries are set up correctly, you can utilize the automatic pad insertion

capabilities provided by Design Analyzer. However, for either case, you will still need to specify an estimate of the expected output load for each output pin.

> **Note:** Automatic pad insertion can cause weird synthesis results when an output pin serves as input to internal logic as well. One way to avoid this is to avoid the use of register type signals as outputs, see Fig. 7.1 and Fig. 7.2 for example code illustrating proper use of registered signals as outputs.

```
//This style will cause pad insertion problems
module bad ( clock, reset, out );
   input clock, reset;
   output out;
    reg out;

  always @(posedge clock) begin
        ...
       out <= ...    // f(out)
        ...
   end
endmodule;
```

Figure 7.1: Bad Style of Coding for Pad Insertion

```
//This style will not cause pad insertion problems
module good ( clock, reset, out );
   input clock, reset;
   output out;
    reg  insideRegister;      // New!

   assign  out = insideRegister;    // New!

  always @(posedge clock) begin
        ...
       insideRegister <= ...    // f(insideRegister)     //New!
        ...
   end
endmodule;
```

Figure 7.2: Good Style of Coding for Pad Insertion

7.8.1 Manual Pad Insertion

By preparing a "wrapper" Verilog file defining the chip-level representation of the design, we can simplify the addition of pads to a synthesized design. For example, our synthesized `converter_gates.v` file shows a top-level module `converter` and the definitions of its inputs and outputs. We can prepare a chip-level module as shown in Fig. 7.3. Note that this is only a fragment of the wrapper file; the full version is available at `/CMC/kits/VRGlocal/demo/converter_chip_wrapper.v`

Run this Unix command to create the file `converter_pads.v,` a complete Verilog description of the chip to be used with other tools:

```
% cat converter_gates.v converter_chip_wrapper.v > converter_pads.v
```

```
module converter_chip ( clk, zero, one, ...  );
  input clk, zero, one, ... ;
  output \segs[7] , \segs[6] , \segs[5] , ... ;
  wire  clk_top, zero_top, one_top, segs_7_top, segs_6_top,
    segs_5_top, ... ;

  -- Insert input pads:
  --   pad-name, instance-name, input-pad-connection, core-wire
  PDIDGZ u_clk  ( .PAD( clk ),          .C(clk_top) );
  PDIDGZ u_zero ( .PAD( zero ),         .C(zero_top) );
  PDIDGZ u_one  ( .PAD( one ),          .C(one_top) );
  ...
  -- Insert output pads:
  --   pad-name, instance-name, core-wire, output-pad-connection
  PDO08CDG u_segs7 ( .I(segs_7_top), .PAD(\segs[7] ) );
  PDO08CDG u_segs6 ( .I(segs_6_top), .PAD(\segs[6] ) );
  PDO08CDG u_segs5 ( .I(segs_5_top), .PAD(\segs[5] ) );
  ...
  -- Instantiate the synthesized module
  converter u_converter (
    .clk (clk_top), .zero (zero_top), .one (one_top), ...
    .\segs[7] (segs_7_top), .\segs[6] (segs_6_top),
    .\segs[5] (segs_5_top), ...
  );
endmodule
```

Figure 7.3: Chip-level module including I/O pads

7.8.2 Automatic Pad Insertion

To automatically insert appropriate input, output, and bidirectional pads into your design[1], you can perform the following steps:

1. Double click on a pin to bring up the *Input/Output Port Attributes* dialog window.
2. Make sure the *Port Is Pad* check-box is selected.
3. Click on **Port Pad Attributes...** button to bring up the *Port Pad Attributes* dialog window.
4. Make sure that the *Build New Pad* radio button is selected. Alternatively, you can specify a specific pad to use as well. Refer to the *Design Analyzer* on-line manual for information about other options in the form.
5. Click **OK** to close the *Port Pad Attributes* window.
6. Click **Apply** to confirm the changes for the selected port.
7. With the *Input/Output Port Attributes* window still open, you can continue changing the attributes for other input or output ports.

Note: You cannot use the *Output Port Attributes* window to set attributes for *input* or *bi-direc-*

1. This works for the CMOSP18 design-kit, but attempts to try this in CMOSP35 designs will cause errors. Input pads will be inserted, but insertion of output pads will fail.

tional ports, and vice versa. You will need to close the current window, and double click on the pin for which you wish to change attributes, to open up its corresponding attribute window.

8. Click **Cancel** to close the *Port Attributes* window.

This has not yet inserted the pads, but only specified ports that should have pads automatically connected to them. To actually insert pads select **Edit→Insert Pads** to bring up the Insert Pads dialog window.

9. Click **OK** in the *Insert Pads* dialog window.

10. Now that you've inserted pads, you will need to re-optimize the design once more to make sure that no timing has been violated. Select **Tools→Design Optimization** to bring up the *Design Optimization* dialog window.

11. Select the *Map Effort* you'd like to use, medium should suffice, and click **OK** to re-optimize your design.

At this stage, you can save your design as **converter_final.v** or **converter_final.vhd**

Also, save the constraints by typing these two commands into the command window:

```
write_constraints -cover_design -format sdf-v2.1 -output converter.sdf
write_sdc converter.sdc
```

7.9 Transfer Synopsys Design to Cadence IC tool via Verilog

1. Start the Cadence IC tool, then select **File→Import→Verilog...**

2. In the pop-up form, specify

   a. *Target Library Name* is the name of an existing Cadence library where you want to import the schematic
   b. *Reference Libraries* should include the names of any standard cell libraries or pad libraries referenced in your Verilog file
   c. *Verilog Files to Import:* indicate the name of your verilog file
   d. Explore the other options; default values should work well
   e. Click **OK** to continue.
   f. On completion, examine the *VerilogIn.log* file for problems.

7.10 Transfer Cadence IC tool Schematic to Synopsys via EDIF

1. Start the Cadence IC tool, select **File→Export→EDIF 200...**

2. In the pop-up form, specify

   a. The *Library, Cell name, View Name* of the schematic to export
   b. *External Libraries* should list the names of standard cell libraries or pad libraries used
   c. *Output File* is the filename to use to save the data

3. In the Synopsys tools, load the EDIF file and save the design.

Chapter 8 Automated Place-and-Route

This chapter demonstrates software that can be used to take a synthesized design from netlist through floor planning, cell placement and automated routing, to create a "correct by construction" chip layout. These tools are very complex, and have many options designers can use to customize their designs; we demonstrate only a very simple flow and leave further investigation of these options to the reader.

The Cadence place-and-route tool is "**EDI**", or "Encounter Digital Implementation", also known as, **Encounter** or "SOC Encounter".

You will need a Verilog gate level netlist of your design as well as design constraint information before beginning floor planning. To demonstrate the required steps, we will continue to use the converter design from the previous chapters.

8.1 Environment Settings

If you have not done so yet, you can set up your design session using the UNIX command

```
% source /CMC/tools/CSHRCs/Cadence.EDI
```

8.2 Required Information

Cadence **Encounter** requires various information before you can use it:

- a gate-level Verilog netlist: We will continue with the example we have been building over the last few chapters, `converter_pads.v`
- information about the technology and the physical libraries you will be using. These are usually set up as *Library Exchange Format* (LEF) files to describe the technology, a standard-cell library, perhaps a separate library of I/O pads, and possibly a library of cells for elimination of antenna effects in wiring. In many cases, these "physical" libraries are proprietary; the companies who supply these often provide us with only a "black-box" representation of the gates within, instead of providing full mask layouts for the gates. The black-box representations supply sufficient information to allow designers and place-and-route tools to use them correctly.
- timing libraries, describing the time delays between inputs and outputs of all of the library cells, and the names of library cells that will be used for buffers and delays
- a file describing the location of input and output pads in your design
- names of power and ground networks
- clock-tree information
- information about constraints

This information can be set up manually, but a template for a specific technology will help. You can create a work directory named **EDI** and copy a sample set-up file and related information using

```
% mkdir EDI
% cd EDI
% cp /CMC/kits/VRGlocal/demo/EDI/CMOSP18/*  .
```

(Note the space followed by the "**.**" at the end of the final command.)

You should see five files in your **EDI** directory:

- converter_pads.v : this is the chip-level Verilog description of the converter design, including pads
- converter.conf : this configuration file for **Encounter** contains much of the set-up, specifying the locations of various files, including the Verilog netlist, the LEF files, the TLF (timing) files, the file describing the location of the I/O pads; it also defines the power and ground networks
- converter.io : the file describing the location of I/O pads
- converter_chip.ctstch : a file describing the clock-tree
- constr.sdc : a file describing constraints

8.2.1 Specifying Pad Locations: converter.io

Unless you are targeting high-density packaging for a system on chip design with hundreds or thousands of pins, the periphery of your chip's floor plan will consist of a single ring of input/output pad circuitry. The pad cells will have power busses running through them to provide power and ground connections for the standard cells in the core of the system ("VDDCORE" and "VSSCORE" networks), and could also have busses running through them to provide separate power and ground connections to the input/output pads ("VDDRING" and "VSSRING" networks). In most modern technologies, I/O pads are designed to be butted against each other -- the LEF (library) information usually does not indicate where these busses run within the I/O cells, so there is no way to automatically route these busses. So, continuity of the I/O ring is important. Pads must abut in order to guarantee continuity of n-wells as well as various metal layers used for I/O and core power distribution. Corner cells provide connectivity for the I/O ring at the corners of the design; their locations must be defined in this file.

There are two types of system designs:

- a ***pad limited*** design has a large number of inputs, outputs, and power and ground connections occupying a majority of the design, with a core (the actual logic cells and wiring) occupying only a small portion of the design;
- a ***core limited*** design has a relatively large core area, with I/O connections not as densely packed as in the previous case

If your design is core limited, the gaps between the pad cells can be filled in by inserting pad filler cells (named **PFEED20**) or by adding additional cells to provide more connections to power or ground. Other constraints imposed by manufacturers (e.g. minimum pitch of bonding pads to ensure that the chip can be packaged properly) may also require that you insert additional filler cells.

Examine the **`converter.io`** file. Each pad is defined using this general template:

```
Pad:  pad_instance_name  side|corner  [cellname]
```

where *`pad_instance_name`* is a unique instance name of a pad, *`side|corner`* can be a compass bearing (N, W, S, or E for normal I/O pads, or NW, SW, NE, SE for corner cells). *`cellname`* is an optional parameter specifying the name of the pad cell from the I/O library.

> **Note:** The Artisan library I/O pad cells do not include the actual pad that will be connected by a wire-bond to the package; the PAD cells (PADIZ40 and PADOZ40) must be added separately, with connections to the "PAD" pin on each I/O pad cell.

8.3 Importing Your Design

1. Start First Encounter by issuing this command in a terminal window:

   ```
   % encounter
   ```

   This should open a graphic window for the tool showing several regions:
   - one large region, which will be used to show a representation of your design
   - top row: banner menu items
   - first row below: icons for some commonly-used functions
   - second row below: icons for various tools
   - far right side of the second row: icons for *floorplan*, *amoeba*, or *physical* views of the design
   - right side at bottom: the "world view" -- showing what area of the design is visible in the large region
   - right side: a region allowing you to change the visibility and selectability of various parts of the design.

   The text terminal where you started **encounter** will display status messages, but can also be used for typing commands.

2. Select **File→Import Design...** from the banner menu of the Encounter widow

3. In the *Design Import* dialog window click **Load...** (at the bottom) and select **converter.conf**, then click **Open**. This reads the configuration file and fills in various fields in the *Design Import* dialog window. Click **OK** to use these files.

The large region should show a floorplan of your design, with pad cells placed as specified in the **converter.io** file. In the middle is the region where the core cells will be placed. Between the core and the I/O ring is a gap, which will be used to route power and ground networks supplying the core cells, as well as for wiring from pad cells to the core circuitry.

Save this initial imported design, using **File→Save Design...** and changing the name to **0_imported.enc** You will be able to return to this initial saved version (or any other saved version) if you find problems in later stages. The steps shown in the remaining parts of this chapter are typical of any design flow. Some steps allow more customization than others; we will only touch on the main issues.

8.4 Initial Floorplan

The initial floorplan is adjustable. You can change various aspects of this floorplan using **Floorplan→Specify Floorplan...** In the *Specify Floorplan* dialog window, you can adjust the aspect ration of the core, the core utilization, the core row spacing and orientation, and many other parameters. After making changes, you can **Apply** them, or **Cancel** any changes.

8.5 Placement of Pre-Built Blocks

In some cases, you might have memories and other pre-designed fixed blocks which have to be placed in the core placement region. Prototype placement is capable of placing macros as well as standard cells in the placement area, but if you wish to do so you may move memories and other fixed macros around, by selecing the block and clicking the **move/resize/reshape** icon in the *tools* list: it looks like a cross with arrow heads.

To decrease the amount of routing congestion around these blocks and to prevent placement of standard cells too close to these blocks it is also a good idea to specify a block halo around them: No cells will be placed inside a block halo. To specify a block halo, select a cell and then select **Floorplan→Edit Floorplan→Edit Halo→...** Fill in the pop-up form appropriately.

8.6 Power Rings and Stripes

Power rings are used to route power busses around the periphery of the core, to make power and ground easily accessible to all areas of the core. For larger designs, where the current in the power and ground nets routed through standard-cell rows can exceed the current-carrying capacity of the metal power and ground connections, power stripes can be run vertically over the rows of standard cells if the metal layer used for these are selected carefully. These ring and stripe traces are thicker and can therefore carry more current. Although, as previously mentioned, a core power ring exists in the pad ring, it is not accessible to the routing engine, and you cannot make connections to it directly except by a connection to

To create a power ring:

1. Select **Power→Power Planning→Add Ring...**

2. In the *Add Rings* dialog window, enter the names of the power nets for which you'd like a ring created (if they're not there already): **VSS VDD**.

3. Select the *ring type*: For a core ring around your placement area, select **core ring(s) contouring** followed by **Around core boundary**. You probably don't want a ring contouring the I/O boundary, as this might cause problems with pad routing connections. (If you have predesigned blocks, you might block rings around these separately.)

4. Specify the ring configuration by specifying the types of metal layers you'd like to use for the different sides of the ring. Specify the width and spacing of the metal layers as well. Default values should be satisfactory for most design, but you may increase the width and spacing of the rings.

5. Choose **center in channel** for the ring offset.

6. Click OK to create the power rings.

7. Select **Power→Power Planning→Add Stripes...** to add vertical stripes across the core region. Try to select the number of stripes and their spacing such that they do not align with power pad cells, to minimize potential problems later. In this example, add one set, offset by 75 microns from the left of the core area.

8. Save the design as **1_power.enc**

8.7 Trial Placement and Routing

At this point we are ready to test the placement of core cells. For complex designs it is advisable to run prototype placement first, before committing CPU resources to running a full blown placement. Prototype placement will give you a sense of how the standard cells will be arranged, so you can get a determine whether or not you need to adjust floorplanning parameters for a more or less aggressive core utilization.

1. Select **Place→Place Standard Cell...**

2. In the *Place* dialog window, choose *Run Placement in Floorplan Mode*: a **prototyping** effort is a good place to start for large designs to determine if the cells can be placed successfully. Use the *Run Full Placement* level for a final placement run later, or for simple designs.

3. Click **OK** to start the placement engine. The core area will contain boxes representing the cells that have been placed. (You might need to change the view to "Physical View", by selecting the icon at the far right side of the *Tools* icon row.)

Run a trial route and simple timing verification to determine if there are any major problems:

4. Select **Route→Trial Route...** and use the default values; click **OK** to continue.

5. Select **Timing→Extract RC** and turn off the **Save Cap to** check-box; click **OK** to continue.

6. Click **Timing→Timing Analysis...**

   In the *Timing Analysis* window, select *Use Existing Extraction and Timing Data*, then click **OK** to continue. Ensure that there are no timing violations. If timing constraints are *not* met, you can try re-doing the **Placement** with a higher *Placement Effort Level*.

7. Run a **Placement** with a *Placement Effort Level* of **High** and re-run the **Trial Route, ExtractRC** and **Timing Analysis**

8. Save this design as **2_placed**

8.8 Clock Tree Insertion

Insertion of a clock tree is an important step in the design of your system, to ensure a perfectly synchronized clock throughout the layout of the design. This is a complex step, but a sample is shown below.

When you started this chapter, you copied several files from the demonstration directory, including the file **converter_chip.ctstch** This clock tree specification file allows you to specify many parameters, including the depth of the clock tree, how many instances there are at each level, and the type and strength of the buffers at each level. For your own designs, you will have to create your own clock tree specification file, but you can use this clock tree specification file as a starting point.

```
AutoCTSRootPin    u_clk/C
MaxDelay           2ns
MinDelay           0ns
SinkMaxTran     100ps
BufMaxTran      150ps
MaxSkew          30ps
NoGating       rising
MaxDepth           7
Buffer           CLKBUFX20 CLKINVX20 CLKBUFX16 CLKINVX16 CLKBUFX12
+ CLKINVX12 CLKBUFXL CLKINVXL CLKBUFX8 CLKINVX8 CLKBUFX4 CLKINVX4
+ CLKBUFX3 CLKINVX3 CLKBUFX2 CLKINVX2 CLKBUFX1 CLKINVX1
End
```

Figure 8.1: Sample Clock-Tree Specification, converter_chip.ctstch

A brief description:

- **AutoCTSRootPin u_clk/C**Root pin specifies the pin where the clock tree begins. Here it is specified by *u_clk*, the instance name of the clock pad, and *C* the port of the pad.
- **MaxDelay 2ns** Maximum delay to tolerate from clock path beginning to end.
- **MinDelay 0ns** Minimum delay to tolerate from clock path beginning to end.
- **MaxSkew 30ps** Maximum clock skew in between different leaf nodes on the clock tree.
- **NoGating rising** Specifies whether clock gating is being used in the design.
- **MaxDepth 7** Depth of the clock tree
- **Buffer CLKBUFX20 CLKINVX20 CLKBUFX16...** Names of buffers to use in the clock tree.
- **End** End the specific file.

You can check to see if this clock timing information was read correctly when you ran the *Design Import*. Use **Timing→Report→Clock Waveform...** and click **OK.** This will show whether a clock signal is recognized from the constraints file.

Now we are ready for clock-tree synthesis. Load the clock-tree specification file and synthesize the tree:

1. Select **Clock→Synthesize Clock Tree...**

2. In the *Specify Clock Tree* dialog window, enter the file name of the clock tree specification file into the provided field: **converter_chip.ctstch**

3. Click **OK** to load the file. Check the terminal window from which you started Encounter for any error messages. You can check to see if this clock timing information was read correctly when you ran the *Design Import*. Use **Timing→Report→Clock Waveform...** and click **OK.** This will show a summary of clocks in your terminal window.Select **Clock→Synthesize Clock Tree...**

4. In the Synthesize Clock Tree dialog window, review the options and click **OK** to generate the clock tree. Clock buffers will be inserted in between placed cells where required in order to meet the timing specifications you provided.

5. Examine the clock-tree using **Clock→Display→Display Clock Tree...**

   Click **Apply** to examine the clock phase delay in the layout
   Change the *Display Selection* by clicking the check-box beside **Display Clock Tree**, then click **OK**
   Change the *Display Selection* to **Display Min/Max Paths** and click **OK**
   Clear the clock display using **Clock→Display→Clear Clock Tree Display**

6. Save your clocked design, **3_clock**.

8.9 The Golden Netlist

The design now contains all of the components that will be used in the final layout. Save this using

Design→Save→Netlist...
Change the name of the file to **converter_chip_gold.v** and click **OK**

8.10 Routing

8.10.1 Route Power Nets

The rows of standard cells have power and ground rails running through them, but these must be connected to each other and to the power rings (and to power stripes, if used). In addition, the I/O pads providing power and ground connections must be connected to the power rails:

1. To complete the power rails in rows where standard cells are used select the *Special Routing* tool, **Route→Special Route...**

2. Click **OK** to complete the power routing. You might see warnings indicating that the design has no block pins to be routed. These are expected, because we have no pre-placed blocks in this example. You may eliminate these warnings by turning off the check-boxes beside "Block Pins" in the *SRoute* dialog window before routing power.

3. Save the design as **4_power**

8.10.2 Routing of Remaining Nets

The final stage of routing, global and detail routing, makes the connections between all of the standard cells and I/O pad cells as indicated in the netlist.

1. Select **Route→NanoRoute→Route...** There are many options in the *NanoRoute* dialog window, which are described fully in the on-line documentation. For this example, we will use the default values, with one exception: Under *Concurrent Routing Features* we will turn on the check-box beside **Timing Driven**

2. Click **OK**

   All of the routing should be complete now.

3. Save the design as **5_routed**

8.11 Place Filler Cells

Note that there are gaps between cells in the standard-cell rows. These cells are typically designed to require abutment with other cells or to be placed on a specific grid to prevent possible design rule violations (e.g. violation of minimum spacing requirements for adjacent n-wells), but forcing a minimum spacing between cells is not practical for area constrained designs. Instead, we make use of **filler** cells -- non-functional blocks providing n-well and/or p-well continuity between cells -- between non-abutted cells, thus eliminating the possibility of design rule spacing violations.

1. Select **Place→Physical Cell→Add Filler...**

2. In the *Add Filler* dialog window, click **Select** to view a list of possible filler cells.

3. To select all of the possible filler cells in the new *Select Filler Cells* dialog window, so wider cells will be used to fill in wide gaps as appropriate, you can select each one individually and click **Add** to make it available for the automated placer, or select all of them at once using this method:

   a. Left-click on the name of the cell at the top of the list
   b. SHIFT-left-click on the name of the cell at the bottom of the list, so all are highlighted
   c. Click **Add**
   d. Click **Close** to proceed

4. In the *Add Filler* dialog window, click **OK.** You might need to refresh the layout window to view the results.

5. Save the design as **6_filler**

8.12 Optional: Metal Fill (if necessary -- CMRF8SF)

Most modern technologies require that the design meets minimum density requirements for metal layers (and possibly others) to ensure that the deposition and patterning of the metals on the wafer during fabrication have a consistent topography. This can be done in the DFII layout environment or here in Encounter.

8.13 Verification

You can verify various aspects of the final design:

1. Select **Verify→Verify Connectivity...**

   This ensures that all connections are correct.

2. Select **Verify→Verify Geometry...**

   This ensures that there are no design rule violations

3. Select **Verify→Verify Process Antenna...**

   During fabrication, metal layers are deposited and etched to provide signal wiring; during etching, electrical charge can build up on these metal layers. When this charge builds up sufficiently, the gate oxide at transistor gates may be damaged, adversely affecting yield.

8.14 Final Timing Analysis

1. Select **Timing→Extract RC** and turn off the **Save Cap to** check-box; click **OK** to continue.

2. Click **Timing→Timing Analysis...**

   In the *Timing Analysis* window, select *Use Existing Extraction and Timing Data* then click **OK** to continue. Ensure that there are no violating paths.

8.15 Saving the Design

1. Save the design as **7_final.enc**

2. ***Optional, possibly required for some technologies like CMRF8SF:***

   In the shell window where you started **encounter**, type

   ```
   set dbgOutLefVias 1
   ```

   to eliminate a bug, where some vias are not written out properly when writing a DEF file

3. Select **Design→Save→DEF...**

   Change the version number to use **5.5** if exporting to Cadence IC5, because that tools does not understand newer versions of the DEF language. Keep the other default values and provide a filename in which to store the *Design Exchange Format* (DEF) file. Click **OK**.

4. Select **Design→Save→GDS...**

   Keep the default values and provide a filename in which to store the Stream/GDS file. Click **OK**. (In order to use this GDS file to import a design into Cadence IC tools, other work will be necessary; that work will not be covered in this manual.

8.16 Importing the Design into Cadence IC tools

8.16.1 Creating a Layout cellview by importing a DEF file

1. Start by creating a new work directory to hold the Cadence IC tools version of this design, and starting a Cadence IC tools session

   ```
   % mkdir Cadence ; cd Cadence
   % startCds -t cmosp18
   ```

2. Ensure that the **cds.lib** file in your work directory includes references to the two cell libraries:

   ```
   DEFINE artisan_sc_30 /CMC/kits/artisan.3.0/FAB/artisan_sc_30
   DEFINE artisan_io_30 /CMC/kits/artisan.3.0/FAB/artisan_io_30
   ```

3. Create a new library, into which you will import the DEF file. To read the DEF correctly, you must (temporarily) attach your library to **cmosp18_defin_techlib** like this:

   Use **CIW→File→New→Library...**
   Library name: **Converter**
   Attach to: **cmosp18_defin_techlib**
   ... and click **OK**

4. Use **CIW→File→Import→DEF...**

   Library Name: **Converter**
   Cell Name:**converter_chip**
   View Name: **autoLayout**
   Use Ref. Library Names:**artisan_sc_30 artisan_io_30**
   DEF File Name: **../converter_chip.def**
   ... and click **OK**
   Upon successful completion, the *Library Manager* should indicate that there exists a cell with the name of your design, with a view *autoLayout*, but **beware: the cell only exists in memory, and has not been save to disk yet!** To save the data, open the *autoLayout* cellview and use

Design→Save.

At this point, the design will have a property associated with it which indicates that the design was created using one of the place-and-route tools, so when you use the IC tools to try to open the design, an old place-and-route tool "takes over", installing its own bindkeys and banner menus. In our design environment, we wish to use the normal Layout tool menus and bindkeys, so we need to remove that "special" property:

5. In the layout window, select **Tools→Layout** to change the menus to those used in the Layout tool.

6. Select **Design→Properties...** and then select the **Property** check-box in the *Edit Cellview Properties* dialog window.

7. Scroll down until you see the property name **viewSubType** with a value of *designPlanC3*. Left-click on the property name (*viewSubType*) to highlight it, then click the **Delete** button at the bottom of the window. Click **OK** to close this window.

8. Save the design, using **Design→Save As...** and save the design with a cellview of ***layout***.

   Note: Spelling and case of the letters is significant!

9. Close the *autoLayout* view of the design

10. Continue by opening the *layout* view of the design.

8.16.2 Creating a Schematic cellview by importing a Verilog file

1. Use **File→Import→Verilog...**

2. In the pop-up form, specify

   a. *Target Library Name*, the name of the library into which you want to import the design
   b. *Reference Libraries* should include **basic** and the names of any standard cell libraries and pad libraries used in the design.
   c. Click **OK** to continue.

CHAPTER 10

Cadence-Synopsys Data Transfer

The recommended method for transferring a design from Synopsys to Cadence is through the Verilog description language. Transfer from Cadence to Synopsys is through EDIF.

10.1 Transfer Schematic from Synopsys to Cadence

1. While running the *Synopsys Design Analyzer*, you can save a schematic in various formats. For a design named *converter*, you can save the `converter.db` schematic in Verilog format under the filename `converter.v` by selecting **File** → **Save As...**.

   a) Enter the output file name, **converter.v**.

   b) Set the *output format* to **Verilog**.

   c) Click **OK** to save the file

   d) Copy file `converter.v` to your Cadence working directory for your current technology, i.e. `~/CMOSP18`.

2. Start Cadence. In the *Cadence CIW*, select **File** → **Import** → **Verilog...**. The *Verilog In* form will appear.

   a) Specify the name of your working library (e.g. test) in the *Target Library Name* field.

b) Add **_wcells** and *padsp35* to the list of libraries in the *Reference Libraries* field, for the *CMOSP35* technology.

Add **vst_n18_sc_tsm_c4** and **tpz973g** to the list of libraries in the *Reference Libraries* field, for the *CMOSP18* technology.

Remove library sample from this list if it exists.

c) Enter the file name of the exported Verilog netlist (**converter.v**) in the *Verilog Files To Import* field.

d) Enter `/CMC/kits/cmosp35/models/verilog/nwb/` in the *-y Options* field, for the *CMOSP35* technology.

Enter `/CMC/kits/cmosp18/VSdir/verilog/vs18sc.v` and `/CMC/kits/cmosp18/VSdir/verilog/tpz973g.v` in the *-v Options* field, for the *CMOSP18* technology.

e) If a design named converter already exists in your test library and you want it overwritten by the design imported from Synopsys, toggle *Overwrite Existing Views* switch **on**.

f) Click **OK** in the *Verilog In* form. Once the conversion is completed a Log File window will pop up. Check that the design was imported successfully. Messages indicating that modules are already in a target/reference library may be ignored. Close the *Log File* window. Close the *Verilog In* form.

g) Select **View** → **Refresh** in the *Library Manager* menu to make the converter cell accessible.

h) You can now edit the imported schematic cellview. Note that in Cadence environment, the symbols have *vdd* and *vss* terminals. Use the *hook up pins routine* (as described in Chapter 4:Schematic Entry) to connect all these terminals. You will get some error messages since the imported design's symbol view created by *Verilog In* routine does not have *vdd* and *vss* pins, whereas the schematic view does. You can delete the symbol view of the converter cell using *Library Manager* and then **Check and Save** the converter schematic. This time there should be no errors. After that, if you need a symbol for the converter cell, you can create it using **Design** → **Create Cellview** → **From Cellview.**

10.2 Transfer Schematic Cellview from Cadence to Synopsys

1. Copy the template file `/CMC/tools/cadence/tools/dfII/samples/xlUI/edifOut.il` into a file like `myEdifOut.il` in your working directory.

2. Select **File** → **Export** → **EDIF 200** in the CIW.

3. Type **myEdifOut.il** in the *Template File* field and click **Load**.

4. Fill some of the subsequent fields as follows:

Library Name	**your library name**
Cell Name	**your cell name**
View Name	**schematic**
External Libraries	for *CMOSP35*: **wcells padsp35**
	for *CMOSP18*: **vst_n18_sc_tsm_c4 tpz973g**
Output File	**filename of your choice**

5. Click **Save** to save the latest settings in `myEdifOut.il`. Click **OK**.

6. The EDIF file `*.edif` (e.g. `converter.edif` is generated. Before importing this file into Synopsys, the EDIF generated from cmosp35[1] designs should be repaired, by removing references to VDD! and VSS! A simple script is available to assist you; use the command
   FixP35Edif converter.edif New_converter.edif which will read the file named in the first argument (`converter.edif`) and write the corrected EDIF into the file named in the second argument (`New_converter.edif`). You can rename the EDIF file:
   mv converter.edif converter_old.edif
   mv New_converter.edif converter.edif

7. Read the exported EDIF file into Synopsys. When you read the EDIF file into Synopsys, save it in VHDL format. You have to comment out the following lines
   `use wcells.COMPONENTS.all;`
   `use padsp35.COMPONENTS.all;`
   in the VHDL file before you use it again in Synopsys or else you will get errors when reading in the VHDL file. This also applies to all VHDL input files.

[1] This might be applicable to CMOSP18 designs as well