

# 6

## Working With Designs in Memory

---

Design Compiler reads designs into memory from design files. Many designs can be in memory at any time. After a design is read in, you can change it in numerous ways, such as grouping or ungrouping its subdesigns or changing subdesign references.

This chapter contains the following sections:

- [Understanding Design Terminology](#)
- [Reading Designs](#)
- [Listing Designs in Memory](#)
- [Setting the Current Design](#)
- [Linking Designs](#)
- [Listing Design Objects](#)
- [Specifying Design Objects](#)

- Creating Designs
- Copying Designs
- Renaming Designs
- Changing the Design Hierarchy
- Editing Designs
- Translating Designs From One Technology to Another
- Removing Designs From Memory
- Saving Designs
- Working With Attributes

---

## Understanding Design Terminology

Different companies might use different terminology for designs and their components. This section describes the terminology used by the Synopsys synthesis tools.

---

### Designs

Designs are circuit descriptions that perform logical functions. Designs are described in various design formats, such as VHDL, Verilog HDL, state machine, and Electronic Data Interchange Format (EDIF).

Logic-level designs are represented as sets of Boolean equations. Gate-level designs, such as netlists, are represented as interconnected cells.

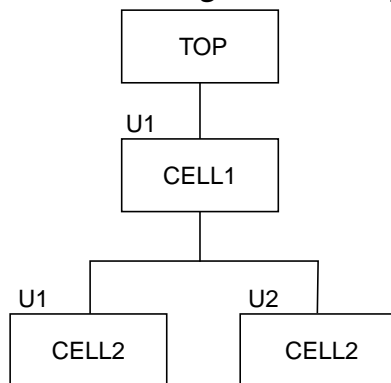
Designs can exist and be compiled independently of one another or can be used as subdesigns in larger designs. Designs are hierarchical or flat.

## Hierarchical Designs

A hierarchical design contains one or more designs as subdesigns. Each subdesign can further contain subdesigns, creating multiple levels of design hierarchy. Designs that contain subdesigns are called parent designs.

Figure 6-1 shows the three levels of hierarchy in the TOP design.

*Figure 6-1 TOP Design Hierarchy*



TOP has three input ports and one output port. The ports of CELL1 and CELL2 are also pins of TOP. TOP and CELL1 are parent designs. CELL2 is instantiated twice in CELL1.

## Flat Designs

Flat designs contain no subdesigns and have only one structural level. They contain only library cells.

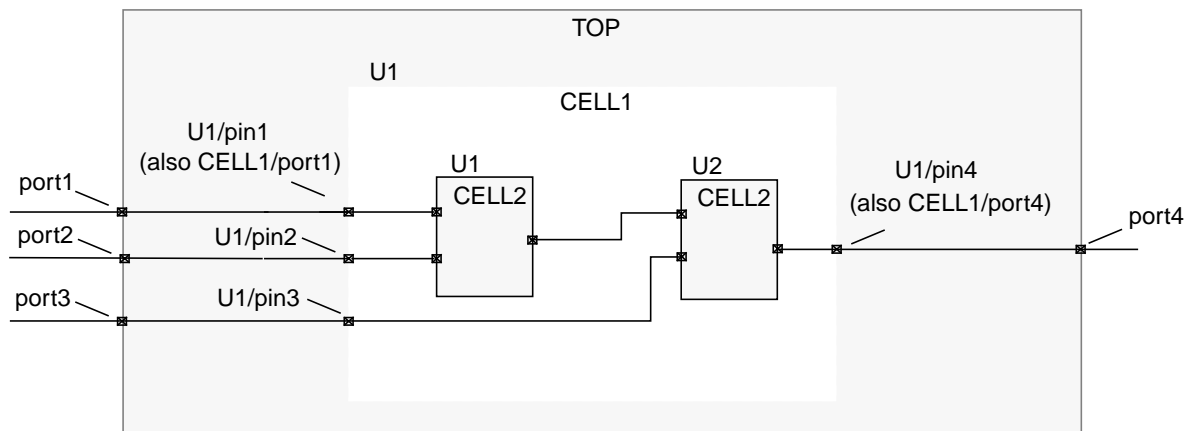
## Design Objects

A design consists of cells, nets, ports, and pins. It can contain subdesigns and references to subdesigns and library cells.

Synopsys commands, attributes, and constraints are directed toward a design object.

Figure 6-2 shows the design objects in the TOP design example.

Figure 6-2 Design Objects in TOP Design



## Current Design

The active design (the design being worked on) is called the current design. Most commands are specific to the current design, that is, they operate within the context of the current design.

## Cells, Instances, and References

A unique instance of a design within another design is called a hierarchical cell. A unique instance of a library cell within a design is called a leaf cell. The term *instance* is also used to refer to a cell.

Some commands work within the context of a hierarchical instance of the current design. The current instance defines the active instance for these instance-specific commands.

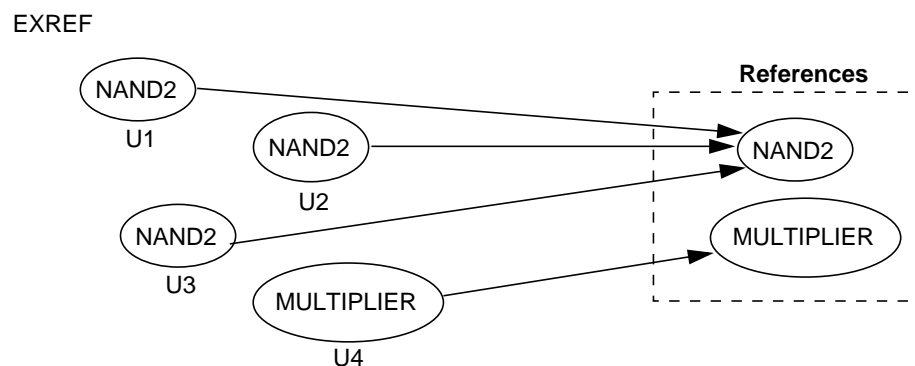
Cells are defined by their attributes, such as their functions, wiring (connections), size, and timing.

A design can contain multiple instances of identical subdesigns or library cells. The instantiated subdesign or library cell is called the reference. Each instance points to the same reference but has a unique name to differentiate it from the other instances.

References enable you to optimize every cell (such as a NAND gate) in a single design without affecting cells in other designs. The cell references in one design are independent of the same cell references in a different design.

Figure 6-3 shows the relationships among designs, cells, and references.

*Figure 6-3 Cells and Cell References*



The EXREF design contains two references: NAND2 and MULTIPLIER. NAND2 is instantiated three times, and MULTIPLIER is instantiated once.

The names given to the three instances of NAND2 are U1, U2, and U3. The references of NAND2 and MULTIPLIER in the EXREF design are independent of the same references in different designs.

For information about resolving references, see “Linking Designs” on page 6-13.

## **Nets**

Nets (networks) are the wires that connect ports to pins and pins to each other. A net is the electrical connection between ports and pins of a signal (from the first port or pin to the last in the signal).

## **Ports**

Ports are the inputs and outputs of a design. The signal flow of ports is defined as input, output, or inout.

## **Pins**

Pins are the input and output of cells within a design (such as gates and flip-flops). The ports of a subdesign are pins within the parent design.

---

## Reading Designs

Table 6-1 lists the design file formats supported by Design Compiler. Design Compiler supports input and output of all formats listed in this table. All formats except .db, EDIF, equation, PLA, and state table require special license keys.

*Table 6-1 Supported Design File Formats*

<b>Format</b>	<b>Description</b>	<b>Keyword</b>	<b>Extension</b>
.db	Synopsys internal database format	db	.db
EDIF	Electronic Design Interchange Format (see the Synopsys <i>EDIF 2 0 0 Interface User Guide</i> )	edif	.edif
equation	Synopsys equation format	equation	.eqn
LSI	LSI Logic Corporation (NDL) netlist format	lsi	.NET
Mentor	Mentor NETED do Format (output only)	mentor	.neted
MIF	Mentor Intermediate Format (input only)	mif	.mif
PLA	Berkeley (Espresso) PLA format	pla	.pla
state table	Synopsys state table format	st	.st
TDL	Tegas Design Language netlist format	tdl	.tdl
Verilog	Verilog Hardware Description Language (see the <i>HDL Compiler for Verilog Reference Manual</i> )	verilog	.v
VHDL	VHSIC Hardware Description Language (see the <i>VHDL Compiler Reference Manual</i> )	vhdl	.vhd
XNF	Xilinx netlist format (see the <i>FPGA Compiler User Guide</i> )	xnf	.xnf

---

Design Compiler provides two ways to read design files:

- The `read_file` command

```
dc_shell> read_file -format keyword design_file
```

- The `analyze` and `elaborate` commands

```
dc_shell> analyze -format keyword design_file
dc_shell> elaborate design_name
```

Table 6-2 summarizes the differences between using the `read_file` command and using the `analyze` and `elaborate` commands to read design files.

*Table 6-2 read\_file Versus analyze and elaborate Commands*

Comparison	<code>read_file</code> Command	<code>analyze</code> and <code>elaborate</code> Commands
Input formats	All formats	VHDL, Verilog
When to use	Netlists, precompiled designs, and so forth	Synthesizing VHDL or Verilog
Design libraries	Cannot store analyzed results except in design library WORK	Can store analyzed results in specified design libraries (use the <code>analyze</code> command option <code>-library</code> or <code>-work</code> )
Generics	Cannot pass parameters (must use directives in HDL)	Allows you to set parameter values on the <code>elaborate</code> command line
Architecture	Cannot specify architecture to be elaborated	Allows you to specify architecture to be elaborated

A design file exists in your host computer's file system. When you read a design file into Design Compiler, it is stored in a memory file in the Synopsys internal database (.db) format. The memory file exists only in the working memory of the Design Compiler program.

Design Compiler names the memory file *path\_name / design.db*. The *path\_name* argument is the directory from which the original file was read, and the *design* argument is the name of the design. If you later read in a design that has the same memory file name, Design Compiler overwrites the original design. To prevent this, use the `-single_file` option with the `read_file` command.

---

## Using a Search Path

You can specify the design file location by using the complete path or only the file name. If you specify only the file name, Design Compiler uses the search path defined in the `search_path` variable to locate the design files. Design Compiler looks for the design files starting with the leftmost directory specified in the `search_path` variable and uses the first library file it finds. When you specify the path, Design Compiler does not use the search path.

To see where Design Compiler finds a file when using the search path, use the `which` command. For example, enter

```
dc_shell> which my_design.db
{/usr/designers/example/my_design.db}
```

---

## Reading .db Files

The version of a `.db` file is the version of Design Compiler that created the file. To read a `.db` file into Design Compiler, the file must have the same or earlier version than the version of Design Compiler you are running.

If you attempt to read in a .db file generated by a Design Compiler version later than the Design Compiler version you are using, an error message appears. The error message provides details about the version mismatch.

---

## Reading HDL Designs

Use the following process to read HDL designs:

1. Analyze the top-level design and all subdesigns in bottom-up order (to satisfy any dependencies).
2. Elaborate the top-level design and any subdesigns that require parameters to be assigned or overwritten.

## Analyzing Designs

The `analyze` command

- Reads an HDL source file
- Checks it for errors (without building generic logic for the design)
- Creates HDL library objects in an HDL-independent intermediate format
- Stores the intermediate files in a location you define

If the `analyze` command reports errors, fix them in the HDL source file and run `analyze` again.

Once a design is analyzed, you must reanalyze the design only when you change it. In addition, because Design Compiler and the Synopsys VHDL System Simulator (VSS) use a common analyzer, you do not need to reanalyze VHDL files analyzed during simulation (unless they have changed).

## Elaborating Designs

The `elaborate` command creates a technology-independent design from the intermediate files produced during analysis. You can override default parameter values during elaboration. Elaboration replaces the HDL arithmetic operators in the code with DesignWare components and determines the correct bus size.

For more information about the `analyze` and `elaborate` commands, see the *HDL Compiler for Verilog Reference Manual* or the *VHDL Compiler Reference Manual*.

---

## Listing Designs in Memory

To list the names of the designs loaded in memory, use the `list_designs` command.

```
dc_shell> list_designs
A (*)      B      C
1
```

The asterisk (\*) next to design A shows that A is the current design.

To list the memory file name information along with the design names, use the `-show_file` option.

```
dc_shell> list_designs -show_file
```

```
/user1/designs/design_A/A.db
A (*)
```

```
/home/designer/dc/B.db
B      C
1
```

The asterisk (\*) next to design A shows that A is the design you are working on. File B.db contains both designs B and C.

To check for duplicate designs loaded in memory, use the `list_duplicate_designs` command.

```
dc_shell> list_duplicate_designs
Warning: Multiple designs in memory with the same design
name.
```

Design	File	Path
-----	----	----
seq2	A.db	/home/designer/dc
seq2	B.db	/home/designer/dc

1

---

## Setting the Current Design

The `current_design` variable points to the current design and is set in the following ways:

- With the `read_file` command

When the `read_file` command successfully completes processing, it sets the current design to the design read in.

```
dc_shell> read_file -format edif MY_DESIGN.edif
Loading edif file '/designs/ex/MY_DESIGN.edif'
```

```
Current design is now '/designs/ex/  
MY_DESIGN.edif:MY_DESIGN'  
{"MY_DESIGN" }
```

- With the `current_design` command

Use this command to set any design in `dc_shell` memory as the current design.

```
dc_shell> current_design ANY_DESIGN  
Current design is 'ANY_DESIGN'.  
{"ANY_DESIGN" }
```

To display the name of the current design, enter

```
dc_shell> list current_design  
current_design = "/usr/home/designs/  
my_design.db:my_design"  
1
```

---

## Linking Designs

For a design to be complete, it must be connected to all the library components and designs it references. For each subdesign, a reference and a link must exist between the subdesign and a design or component in the link libraries. This process is called linking the design or resolving references.

Design Compiler resolves references by

- Determining which library components and subdesigns are referenced in the current design and its hierarchy
- Searching the link libraries to locate these references

Design Compiler first searches the libraries and design files defined in the current design's `local_link_library` attribute, then searches the libraries and design files defined in the `link_library` variable. See Chapter 5, "Working With Libraries," for more information about link libraries.

**Note:**

In a hierarchical design, Design Compiler considers only the top-level design's local link library. It ignores local link libraries associated with the subdesigns.

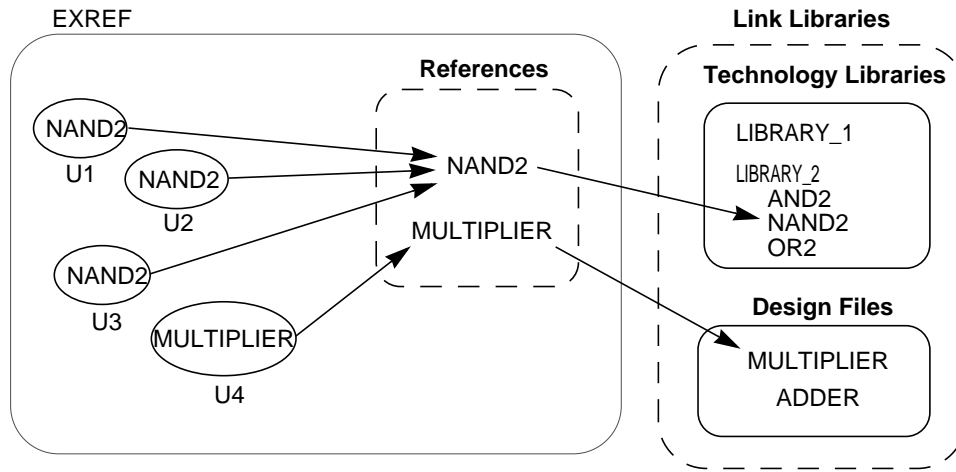
Design Compiler uses the first reference it locates. If it locates additional references with the same name, it generates a warning message identifying the ignored, duplicate references. If Design Compiler does not find the reference, a warning appears advising that the reference cannot be resolved.

- Connecting the located references to the design

By default, the case sensitivity of the link process depends on the source of the references. To explicitly define the case sensitivity of the link process, set the `link_force_case` variable.

The arrows in Figure 6-4 show the connections the linking process added between the cells, references, and link libraries. In this example, Design Compiler finds library component NAND2 in the LIBRARY\_2 technology library; it finds subdesign MULTIPLIER in a design file.

Figure 6-4 Resolving References



You can link the design either manually or automatically. The following sections describe these tasks.

---

## Linking a Design Manually

Use the `link` command to manually link a design. The `link` command removes existing links before starting the link process.

---

## Linking a Design Automatically

The following `dc_shell` commands automatically link designs:

- `compile`
- `create_schematic`
- `group`
- `check_design`
- `report`

- `compare_design`

When Design Compiler links automatically, it does not remove existing links. The automatic link process works only on unlinked components and subdesigns.

---

## Changing Design References

Use the `change_link` command to change the component or design to which a cell or reference is linked.

- For a cell, the link for that cell is changed.
- For a reference, the link is changed for all cells having that reference.

The link can be changed only to a component or design that has the same number of ports with the same size and direction as the original reference.

When you use `change_link`, all link information is copied from the old design to the new design. If the old design is a synthetic module, all attributes of the old synthetic module are moved to the new link. After using `change_link`, manually link the design with the link command.

---

## Listing Design Objects

Design Compiler provides commands for accessing various design objects. These commands refer to design objects located in the current design. Each command performs one of the following actions:

- List

Provides a listing with minimal information.

- Display

Provides a report including characteristics of the design object.

- Return

Returns a list that can be used as input to another `dc_shell` command.

Table 6-3 lists these commands and the actions they perform.

*Table 6-3 Commands to Access Design Objects*

<b>Object</b>	<b>Command</b>	<b>Action</b>
Instance	<code>list_instances</code> <code>report_cell</code>	Lists instances and their references. Displays information about instances.
Reference	<code>report_reference</code>	Displays information about references.
Port	<code>report_port</code> <code>report_bus</code>  <code>all_inputs</code> <code>all_outputs</code>	Displays information about ports. Displays information about bused ports. Returns all input ports. Returns all output ports.
Net	<code>report_net</code> <code>report_bus</code>	Displays information about nets. Displays information about bused nets.
Clock	<code>report_clock</code> <code>all_clocks</code>	Displays information about clocks. Returns all clocks.

*Table 6-3 Commands to Access Design Objects (continued)*

Object	Command	Action
Register	all_registers	Returns all registers.

## Specifying Design Objects

You can specify design objects by using either a relative path or an absolute path.

### Using a Relative Path

If you use a relative path to specify a design object, the object must be in the current design. Specify the path relative to the current instance. The current instance is the frame of reference within the current design. By default, the current instance is the top level of the current design. Use the `current_instance` command to change the current instance.

For example, to place a `dont_touch` attribute on hierarchical cell U1/U15 in the `Count_16` design, you can enter either

```
dc_shell> current_design Count_16
Current design is 'Count_16'.
{"Count_16"}
dc_shell> set_dont_touch U1/U15
```

or

```
dc_shell> current_design Count_16
Current design is 'Count_16'.
```

```
{"Count_16"}
dc_shell> current_instance U1
Current instance is '/Count_16/U1'.
"/Count_16/U1"
dc_shell> set_dont_touch U15
```

In the first command sequence, the frame of reference remains at the top level of design Count\_16. In the second command sequence, the frame of reference changes to instance U1. Design Compiler interprets all future object specifications relative to instance U1.

To reset the current instance to the top level of the current design, enter the `current_instance` command without an argument.

```
dc_shell> current_instance
```

The `current_instance` variable points to the current instance. The `current_reference` variable points to the reference of the current instance. To display the current instance and its reference, enter

```
dc_shell> list current_instance
current_instance = "Count_16/U1"
1
dc_shell> list current_reference
current_reference = "/usr/designs/Count_16.db:Count_4"
1
```

---

## Using an Absolute Path

When you use an absolute path to specify a design object, the object can be in any design in `dc_shell` memory. Use the following syntax when specifying an object by using an absolute path:

[*file* :] *design/object*

### *file*

The path name of a memory file followed by a colon (:). Use the file argument when multiple designs in memory have the same name.

### *design*

The name of a design in dc\_shell memory.

### *object*

The name of the design object, including its hierarchical path. If several objects of different types have the same name, Design Compiler uses the default search order unless you specify the object type.

To specify an object type, use the `find` command. For more information about the `find` command, see the *Design Compiler Command-Line Interface Guide*.

For example, to place a `dont_touch` attribute on hierarchical cell U1/U15 in the `Count_16` design, enter

```
dc_shell> set_dont_touch \  
/usr/designs/Count_16.db:Count_16/U1/U5
```

---

## Creating Designs

The `create_design` command creates a new design. The memory file name is *design.db*, and the path is the current working directory.

```
dc_shell> create_design my_design  
Creating design 'my_design' in file 'my_design.db'.  
1
```

```
dc_shell> list_designs -show_file

/designs/A.db
A (*)

/designs/B.db
B

/usr/work/my_design.db
my_design
1
```

Designs created with `create_design` contain no design objects. Use the appropriate create commands (such as `create_clock`, `create_cell`, or `create_port`) to add design objects to the new design. See “Editing Designs” on page 6-29 for information about these commands.

---

## Copying Designs

The `copy_design` command copies a design in memory and renames the copy. The new design has the same path and memory file as the original design.

```
dc_shell> copy_design A A_NEW
Copying design 'A' to 'A_NEW'
1
dc_shell> list_designs -show_file

/designs/A.db
A      A_NEW

/designs/B.db
B

1
```

You can use the `copy_design` command with the `change_link` command to manually create unique instances. For example, assume that a design has two identical cells, U1 and U2, both linked to COMP. Enter the following commands to create unique instances:

```
dc_shell> copy_design COMP COMP1
Performing copy_design on design 'COMP'.
Copying design 'COMP' to 'COMP1'
1
dc_shell> change_link U1 COMP1
Performing change_link on cell 'U1'.
1
dc_shell> copy_design COMP COMP2
Performing copy_design on design 'COMP'.
Copying design 'COMP' to 'COMP2'
1
dc_shell> change_link U2 COMP2
Performing change_link on cell 'U2'.
1
```

---

## Renaming Designs

The `rename_design` command renames a design in memory.

```
dc_shell> list_designs -show_file

/designs/X.db
A      B
1
dc_shell> rename_design A A_NEW
Moving design 'A' to 'A_NEW'
1
dc_shell> list_designs -show_file

/designs/X.db
A_NEW  B
1
```

**Note:**

Renaming designs might cause unresolved references during linking.

---

## Changing the Design Hierarchy

When possible, reflect the design partitioning in your HDL description. If your HDL code is already developed, Design Compiler allows you to change the hierarchy without modifying the HDL description.

The `report_hierarchy` command displays the design hierarchy. Use this command to understand the current hierarchy before making changes and to verify the hierarchy changes.

Design Compiler provides the following hierarchy manipulation capabilities:

- Adding levels of hierarchy
- Removing levels of hierarchy
- Merging cells from different subdesigns

The following sections describe these capabilities.

---

### Adding Levels of Hierarchy

Adding a level of hierarchy is called grouping. You can create a level of hierarchy by grouping cells or related components into subdesigns.

## Grouping Cells Into Subdesigns

The `group` command groups cells (instances) in the design into a new subdesign, creating a new level of hierarchy. The grouped cells are replaced by a new instance (cell) that references the new subdesign.

The ports of the new subdesign are named after the nets to which they are connected in the design. The direction of each port of the new subdesign is determined from the pins of the corresponding net.

To create a new subdesign,

- Specify the cells included in the new subdesign as the command-line argument.

All cells must be children of the current instance. You can exclude cells from the specified list by using the `-except` option.

- Specify the name of the new subdesign, using the `-design_name` option.
- Specify the new instance name, using the `-cell_name` option. (optional)

If you do not specify an instance name, Design Compiler creates one for you. The created instance name has the format `Un`, where `n` is an unused cell number (for example, `U107`).

For example, to group three cells into a new design named `sample`, enter

```
dc_shell> group {cell1, cell2, cell3} -design_name sample
```

To group all cells that begin with `alu` into a new design `uP` with cell name `UCELL`, enter

```
dc_shell> group "alu*" -design_name uP -cell_name UCELL
```

## Grouping Related Components Into Subdesigns

You also use the `group` command (but with different options) to group related components into subdesigns.

To group related components,

- Specify the component type, using one of the options shown in Table 6-4.

*Table 6-4 Component Grouping Options*

Component	Options
Bused gates	-hdl_bussed
Combinational logic	-logic
Finite state machines	-fsm
HDL blocks	-hdl_all_blocks -hdl_block <i>block_name</i>
PLA specifications	-pla

- Specify the name of the new subdesign, using the `-design_name` option.
- Specify the new instance name, using the `-cell_name` option (optional).

If you do not specify an instance name, Design Compiler creates one for you. The created instance name has the format  $U_n$ , where  $n$  is an unused cell number (for example, U107).

For example, to group all cells in the HDL function bar in the process ftj into design new\_block, enter

```
dc_shell> group -hdl_block ftj/bar -design_name new_block
```

To group all bused gates beneath process ftj into separate levels of hierarchy, enter

```
dc_shell> group -hdl_block ftj -hdl_bussed
```

---

## Removing Levels of Hierarchy

Removing a level of hierarchy is called ungrouping. Ungrouping removes (or collapses) the level of hierarchy of the identified subdesign and merges the subdesign with the surrounding logic.

You can ungroup designs in two ways:

- Immediately, using the `ungroup` command to directly ungroup designs
- During optimization, using the `set_ungroup` command or using the `-ungroup_all` option when you run the `compile` command

Designs that have the `dont_touch` attribute cannot be ungrouped.

## Ungrouping Designs Directly

The `ungroup` command immediately ungroups one or more designs.

To ungroup a design,

- Specify the cells to be ungrouped as the command-line argument.

All cells must be children of the current instance. To ungroup all hierarchical children of the current instance, specify the `-all` option instead of providing a cell list.

By default, the `ungroup` command ungroups only one level of hierarchy in each cell. To ungroup each cell recursively until all levels of hierarchy are removed, specify the `-flatten` option.

- Specify the prefix for the ungrouped cells (optional).

If you do not specify a prefix, Design Compiler uses the prefix *old\_cell\_name/*. When you use the `-flatten` option, do not specify a prefix, because the default cell names are more descriptive than cell names generated with a specified prefix.

If the specified or default prefix creates a nonunique name, Design Compiler adds a number to the end of the cell name to make it unique.

For example, to ungroup several cells, enter

```
dc_shell> ungroup {high_decoder_cell, low_decoder_cell}
```

To ungroup the cell U1 and specify the prefix to use when creating new cells, enter

```
dc_shell> ungroup U1 -prefix "U1_"
```

To completely flatten the current design, enter

```
dc_shell> ungroup -all -flatten
```

## Ungrouping Designs During Optimization

To remove all levels of hierarchy during optimization (including DesignWare parts), use the `-ungroup_all` option when you run the `compile` command.

```
dc_shell> compile -ungroup_all
```

To ungroup specific cells or designs, use the `set_ungroup` command before running the `compile` command. The `set_ungroup` command sets the `ungroup` attribute on the specified cells or designs. If you set the `ungroup` attribute on a cell, during optimization Design Compiler ungroups that cell. If you set the `ungroup` attribute on a design, during optimization Design Compiler ungroups all cells that reference that design.

For example, to ungroup cell U1 during optimization, enter the following commands:

```
dc_shell> set_ungroup U1
dc_shell> compile
```

To see whether an object has the `ungroup` attribute set, use the `get_attribute` command.

```
dc_shell> get_attribute object ungroup
```

To remove an `ungroup` attribute, use the `remove_attribute` command or set the `ungroup` attribute to `false`.

```
dc_shell> set_ungroup object false
```

---

## Merging Cells From Different Subdesigns

To merge cells from different subdesigns into a new subdesign,

1. Group the cells into a new design.
2. Ungroup the new design.

For example, this command sequence creates a new design, `alu`, that contains the cells that initially were in subdesigns `u_add` and `u_mult`.

```
dc_shell> group {u_add, u_mult} -design alu
dc_shell> current_design = alu
dc_shell> ungroup -all
dc_shell> current_design = top_design
```

---

## Editing Designs

Design Compiler provides commands for incrementally editing a design that is in memory. These commands allow you to change the netlist or edit designs by using `dc_shell` commands instead of an external format.

*Table 6-5 Commands to Edit Designs*

Object	Task	Command
Cells	Create a cell	<code>create_cell</code>
	Delete a cell	<code>remove_cell</code>
Nets	Create a net	<code>create_net</code>
	Connect a net	<code>connect_net</code>
	Disconnect a net	<code>disconnect_net</code>
	Delete a net	<code>remove_net</code>

*Table 6-5 Commands to Edit Designs (continued)*

Object	Task	Command
Ports	Create a port	create_port
	Delete a port	remove_port
Buses	Create a bus	create_bus
	Delete a bus	remove_bus

When connecting or disconnecting nets, use the `all_connected` command to see the objects that are connected to a net, port, or pin.

For example, this sequence of `dc_shell` commands replaces the reference for cell U8 with a high-power inverter.

```
dc_shell> find(pin, U8/*)
{"U8/A", "U8/Z"}
dc_shell> all_connected U8/A
{"n66"}
dc_shell> all_connected U8/Z
{"OUTBUS[10]"}
dc_shell> remove_cell U8
Removing cell 'U8' in design 'top'.
1
dc_shell> create_cell U8 IVP
Creating cell 'U8' in design 'top'.
1
dc_shell> connect_net n66 find(pin,U8/A)
Connecting net 'n66' to pin 'U8/A'.
1
dc_shell> connect_net OUTBUS[10] find(pin,U8/Z)
Connecting net 'OUTBUS[10]' to pin 'U8/Z'.
1
```

---

## Translating Designs From One Technology to Another

The `translate` command translates a design from one technology to another.

Designs are translated cell by cell from the original technology library to a new technology library, preserving the gate structure of the original design. The translator uses the functional description of each existing cell (component) to determine the matching component in the new technology library (target library). If no exact replacement exists for a component, it is remapped with components from the target library.

You can influence the replacement-cell selection by preferring or disabling specific library cells (`set_prefer` and `set_dont_use` commands) and by specifying the types of registers (`set_register_type` command). The target libraries are specified in the `target_library` variable. The `local_link_library` of the top-level design is set to the `target_library` value after the design is linked.

The `translate` command does not operate on cells or designs having the `dont_touch` attribute. After the translation process, Design Compiler reports cells that are not successfully translated. During the verification phase, Design Compiler applies the `compare_design` script.

---

### Procedure to Translate Designs

This procedure works for most designs, but manual intervention might be necessary for some complex designs.

To translate a design,

1. Read in your mapped design.

```
dc_shell> read_file design.db
```

2. Set the target library to the new technology library.

```
dc_shell> target_library = { target_lib.db }
```

3. Set timing constraints based on the current design performance.

```
dc_shell> derive_timing_constraints
```

4. Invoke the `translate` command.

```
dc_shell> translate
```

After a design is translated, you can optimize it (using the `compile` command) to improve the implementation in the new technology library.

---

## Restrictions When Translating Between Technologies

Keep these restrictions in mind when you translate a design from one technology to another:

- The `translate` command translates functionality logically but does not preserve drive strength during translation. It always uses the lowest drive strength version of a cell, which might produce a netlist with violations.
- When you translate CMOS three-state cells into FPGA, functional equivalents between the technologies might not exist.

- Buses driven by CMOS three-state components must be fully decoded (Design Compiler can assume that only one bus driver is ever active). If this is the case, bus drivers are translated into control logic. To enable this feature, set the `compile_assume_fully_decoded_three_state_buses` variable to true before translating.
- If a three-state bus within a design is connected to one or more output ports, translating the bus to a multiplexed signal changes the port functionality. Because `translate` does not change port functionality, this case is reported as a translation error.

---

## Removing Designs From Memory

The `remove_design` command removes designs from `dc_shell` memory. For example, after completing a compilation session and saving the optimized design, you can use `remove_design` to delete the design from memory before reading in another design.

By default, the `remove_design` command removes only the specified design. To remove its subdesigns, specify the `-hierarchy` option. To remove all designs (and libraries) from memory, specify the `-all` option.

If you defined variables that reference design objects, Design Compiler removes these references when you remove the design from memory. This prevents future commands from attempting to operate on nonexistent design objects. For example,

```
dc_shell> PORTS = all_inputs()
{"A0", "A1", "A2", "A3"}
dc_shell> list PORTS
PORTS = {"A0", "A1", "A2", "A3"}
dc_shell> remove_design
```

```
Removing design 'top'  
1  
dc_shell> list PORTS  
PORTS = {}
```

---

## Saving Designs

You can save (write to disk) the design and its subdesigns in the hierarchy at any time and to a different name or format. After you modify a design, you must save that design. Design Compiler does not automatically save designs when you exit.

Table 6-6 on page 6-37 lists the design file formats supported by Design Compiler. All formats except .db, EDIF, equation, PLA, and state table require special license keys.

The `write` command converts designs in memory to a format you specify and saves that representation to disk. By default, Design Compiler saves the current design in .db format to the file `./ design_name.db`.

```
dc_shell> write
```

When writing to formats other than .db, consider the naming requirements of the target environment. You might have to perform one or more of the following tasks before saving the design:

- If the target environment has restrictions on the design object names, use the `change_names` command to modify the names.
- If the target environment has specific requirements for bus delimiters, set the `bus_naming_style` variable to meet those requirements.

- If the target environment requires schematics, use the `create_schematic` command to generate the schematics.

To output in another format, use the `-format` option to specify the format.

```
dc_shell> write -format keyword
```

To write a hierarchical design and its subdesigns, specify only the top-level design; do not specify all the design's files. By default, Design Compiler writes each design to a separate file.

```
dc_shell> write -hierarchy top_design
```

To save multiple designs to a single output file, use the `-output` option to specify the output file.

```
dc_shell> write -output file_name design_list  
dc_shell> write -output file_name -hierarchy
```

To save all modified designs to their default files in `.db` format, enter

```
dc_shell> write -modified find( design, "*" )
```

The following special cases apply:

- Synopsys database (`.db`) format is the only output format that can have designs containing unmapped synthetic library cells.
- The EDIF, LSI, and Mentor formats require a mapped design.
- The equation format requires a combinational design.
- Schematics are ignored by equation, LSI, PLA, state table, TDL, Verilog, and VHDL formats.
- The Mentor format requires schematics.

## Working With Attributes

Attributes describe logical, electrical, physical, and other properties of objects in the design database. An attribute is attached to a design object and is saved with the design database.

Design Compiler uses attributes on the following types of objects:

- Entire designs
- Design objects, such as clocks, nets, pins, and ports
- Design references and cell instances within a design
- Technology libraries, library cells, and cell pins

An attribute has a name, a type, and a value. Attributes can have the following types: string, numeric, or logical (Boolean).

Some attributes are predefined and are recognized by Design Compiler; other attributes are user-defined. Appendix C lists the predefined attributes.

Some attributes are read-only. Design Compiler sets these attribute values and you cannot change them. Other attributes are read/write. You can change these attribute values at any time.

Most attributes apply to one object type; for example, the `rise_drive` attribute applies only to input and inout ports. Some attributes apply to several object types; for example, the `dont_touch`

attribute can apply to a net, cell, port, reference, or design. You can get detailed information about the predefined attributes that apply to each object type by using the commands listed in Table 6-6.

*Table 6-6 Commands to Get Attribute Descriptions*

<b>Object Type</b>	<b>Command</b>
All	help attributes
Designs	help design_attributes
Cells	help cell_attributes
Clocks	help clock_attributes
Nets	help net_attributes
Pins	help pin_attributes
Ports	help port_attributes
Libraries	help library_attributes
Library cells	help library_cell_attributes
References	help reference_attributes

---

## Setting Attribute Values

To set the value of an attribute, use

- An attribute-specific command

Use an attribute-specific command to set the value of its associated attribute.

For example,

```
dc_shell> set_dont_touch U1
```

- The `set_attribute` command

Use this command to set the value of any attribute or to define a new attribute and set its value.

For example, to set the `flatten` attribute to `false` on design top, enter

```
dc_shell> set_attribute top flatten false
```

If an attribute applies to more than one object type, Design Compiler searches the database for the named object. See “Understanding the Object Search Order” on page 6-41 for information about the search order.

When you set an attribute on a reference (subdesign or library cell), the attribute applies to all cells in the design with that reference. When you set an attribute on an instance (cell, net, or pin), the attribute overrides any attribute inherited from its reference.

---

## Viewing Attribute Values

To see all attributes on an object, use the `report_attribute` command.

```
dc_shell> report_attribute -object obj_type
```

To see the value of a specific attribute on an object, use the `get_attribute` command.

For example, to get the value of the maximum fanout on port OUT7, enter

```
dc_shell> get_attribute OUT7 max_fanout
Performing get_attribute on port 'OUT7'.
```

```
{3.000000}
```

If an attribute applies to more than one object type, Design Compiler searches the database for the named object. See “Understanding the Object Search Order” on page 6-41 for information about the search order.

---

## Saving Attribute Values

Design Compiler does not automatically save attribute values when you exit `dc_shell`. Use the `write_script` command to generate a `dc_shell` script that recreates the attribute values.

### Note:

The `write_script` command does not support user-defined attributes.

By default, `write_script` prints to the screen. Use the redirection operator (`>`) to redirect the output to a file.

```
dc_shell> write_script > attr.scr
```

---

## Defining Attributes

The `set_attribute` command enables you to create new attributes. Use the `set_attribute` command described in “Setting Attribute Values” on page 6-37.

If you want to change the value type of an attribute, remove the attribute and then re-create it to store the desired type.

## Removing Attributes

To remove a specific attribute from an object, use the `remove_attribute` command.

You cannot use the `remove_attribute` command to remove inherited attributes. For example, if a `dont_touch` attribute is assigned to a reference, remove the attribute from the reference, not from the cells that inherited the attribute.

For example, to remove the `max_fanout` attribute from the port `OUT7`, enter

```
dc_shell> remove_attribute OUT7 max_fanout
Performing remove_attribute on port 'OUT7'.
{OUT7}
```

You can remove selected attributes by using the commands that set them. Some `set_*` commands provide a `-default` option that removes from the current design the attributes previously set by the command. See the man page for a specific command to determine whether it has the `-default` option.

To remove all attributes from the current design, use the `reset_design` command.

```
dc_shell> reset_design
Resetting current design 'EXAMPLE'.
1
```

The `reset_design` command removes all design information, including clocks, input and output delays, path groups, operating conditions, timing ranges, and wire load models. The result of using `reset_design` is often equivalent to starting the design process from the beginning.

## Understanding the Object Search Order

Design Compiler searches for an object, *X*, in the following order:

1. Design *X*
2. Cell *X*
3. Net *X*
4. Reference *X*
5. Library cell *X*

Commands that can set an attribute on more than one type of object use this search order to determine the object to which the attribute applies.

For example, the `set_dont_touch` command operates on cells, nets, references, and library cells. If you define an object, *X*, with the `set_dont_touch` command and two objects (such as the design and a cell) are named *X*, Design Compiler applies the attribute to the first object type found. In this case, the attribute is set on the design, not on the cell.

Design Compiler stops searching when it finds a matching object, or it displays an error message if it does not find a matching object.

Design Compiler echoes the type of object on which an attribute is set. (If you do not want the echo, set `verbose_messages = false`.)

```
dc_shell> set_dont_touch X
Performing set_dont_touch on design 'X'.
1
```

You can override the default search order by using the `find` command to specify the object.

For example, assume that the current design contains both a cell and a net named `critical`. The first command sets the `dont_touch` attribute on the cell because of the default search order; the second command places the `dont_touch` attribute on the net.

```
dc_shell> set_dont_touch critical  
Performing set_dont_touch on cell 'critical'.  
1  
dc_shell> set_dont_touch find(net, critical)  
Performing set_dont_touch on net 'critical'.  
1
```