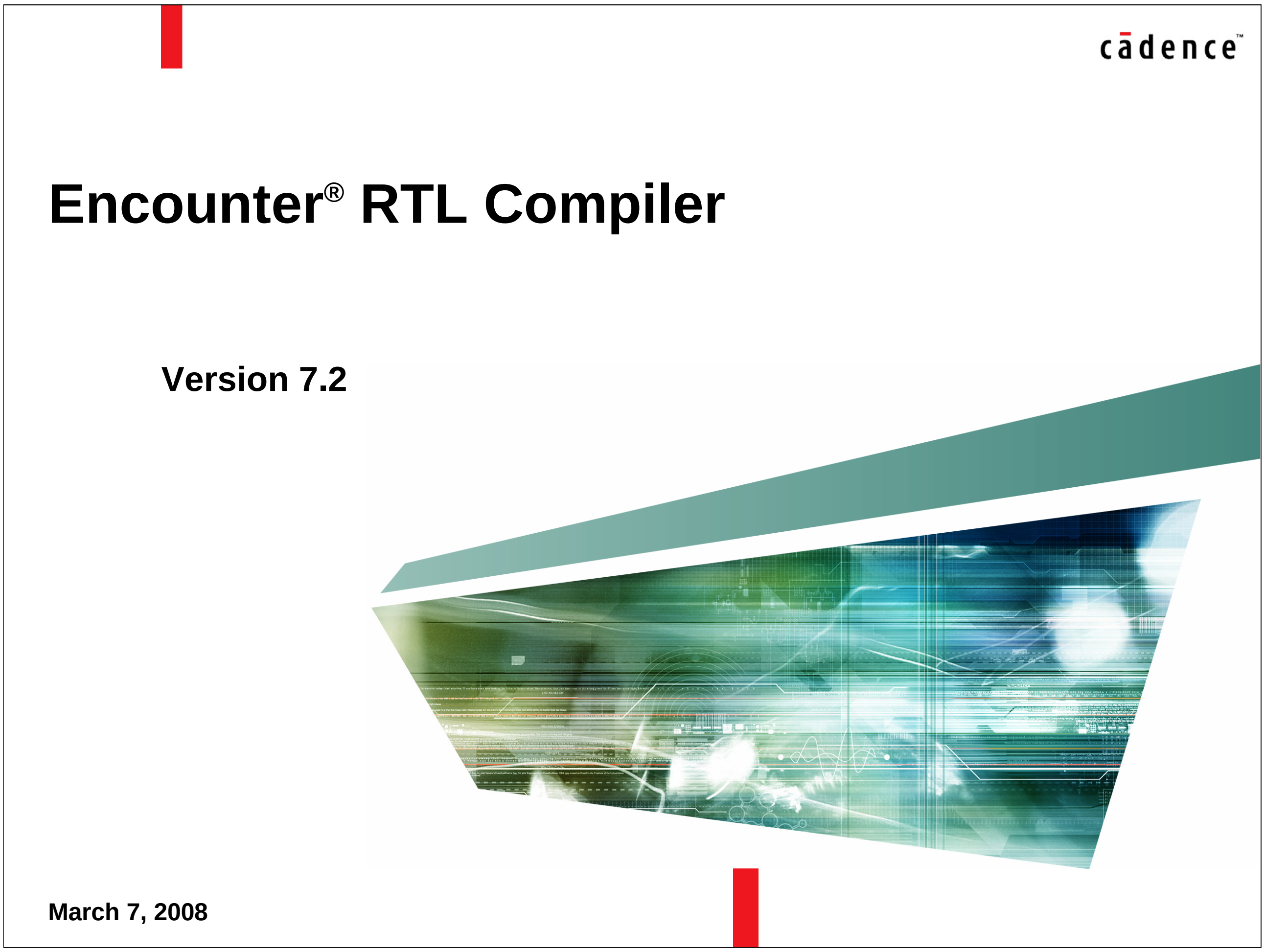
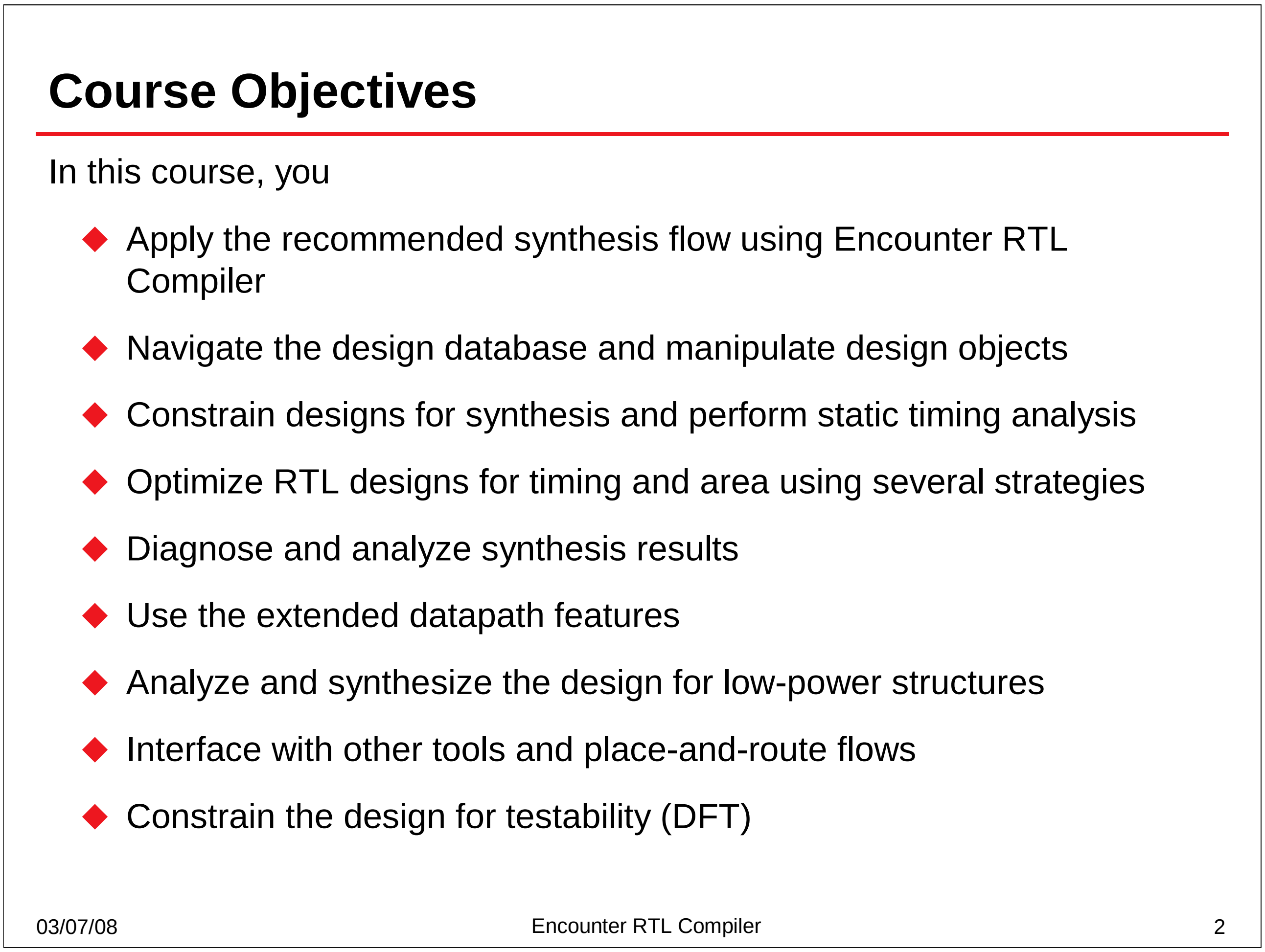


cādence

Encounter® RTL Compiler

Version 7.2

March 7, 2008

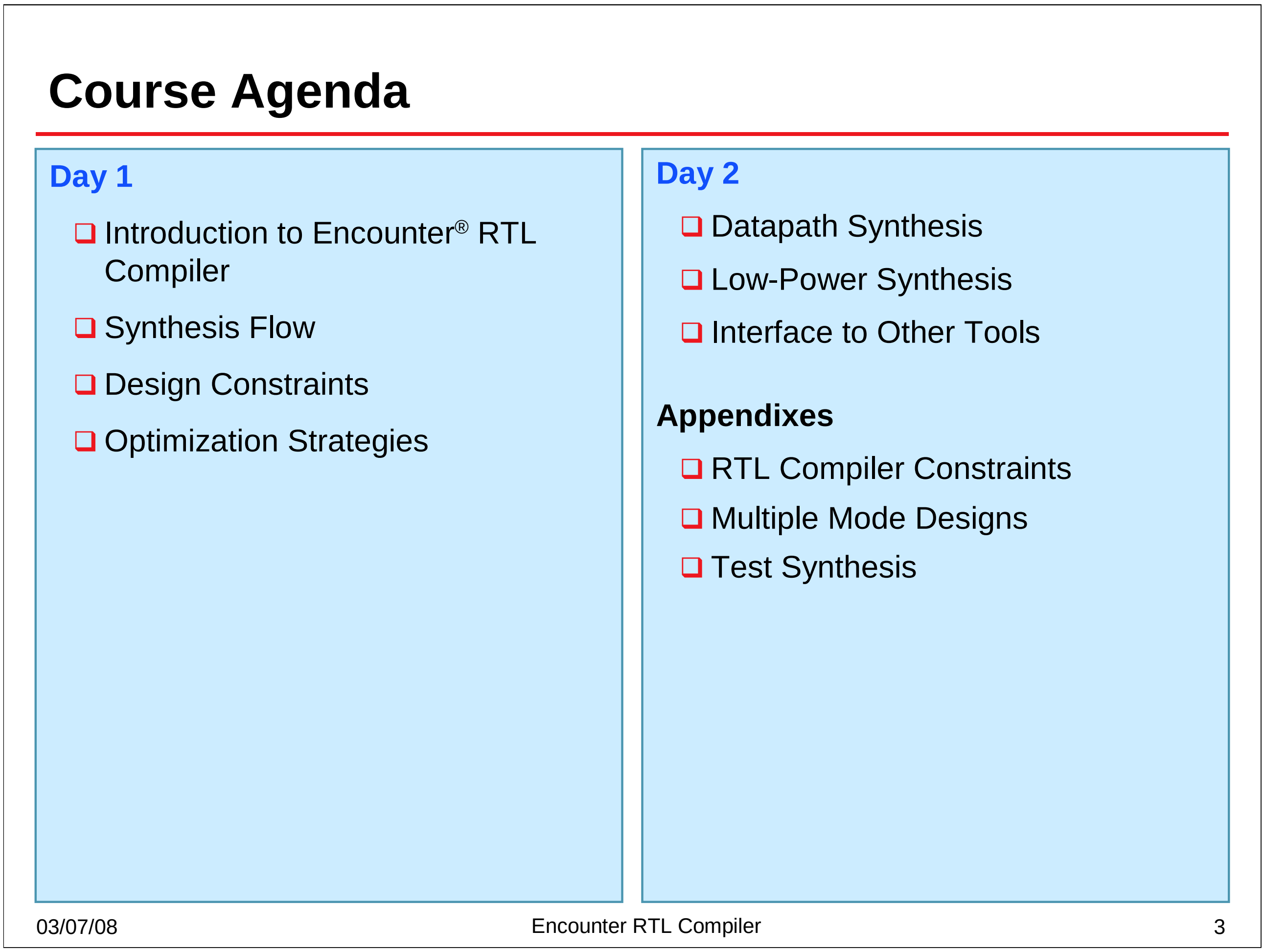


Course Objectives

In this course, you

- Apply the recommended synthesis flow using Encounter RTL Compiler
- Navigate the design database and manipulate design objects
- Constrain designs for synthesis and perform static timing analysis
- Optimize RTL designs for timing and area using several strategies
- Diagnose and analyze synthesis results
- Use the extended datapath features
- Analyze and synthesize the design for low-power structures
- Interface with other tools and place-and-route flows
- Constrain the design for testability (DFT)

03/07/08 Encounter RTL Compiler 2



Course Agenda

Day 1

- Introduction to Encounter® RTL Compiler
- Synthesis Flow
- Design Constraints
- Optimization Strategies

Day 2

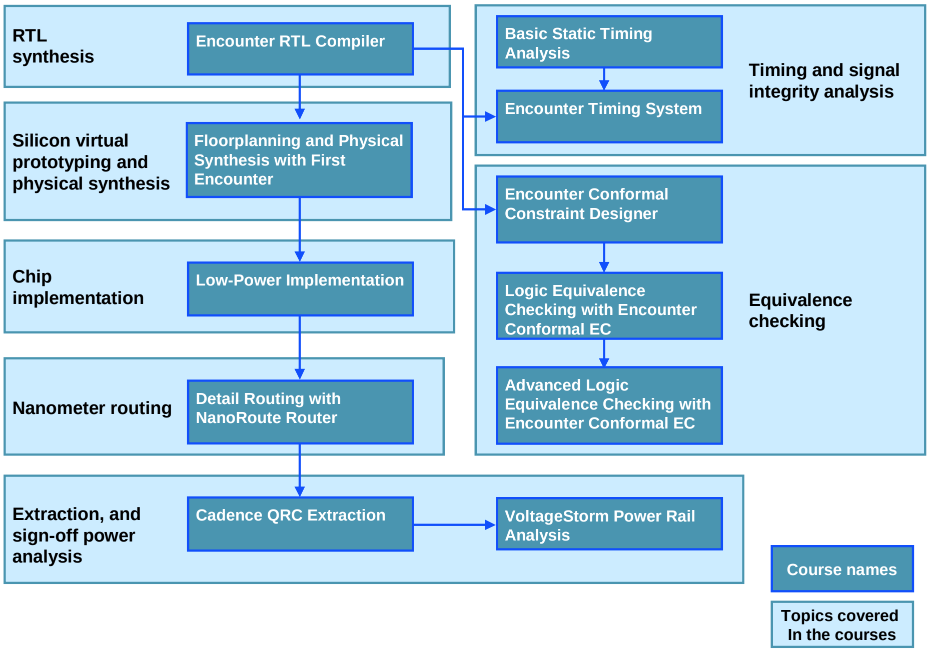
- Datapath Synthesis
- Low-Power Synthesis
- Interface to Other Tools

Appendixes

- RTL Compiler Constraints
- Multiple Mode Designs
- Test Synthesis

Curriculum Planning

Cadence offers courses that are closely related to this one. You can use this course map to plan your future curriculum.



RTL synthesis

Encounter RTL Compiler

Silicon virtual prototyping and physical synthesis

Floorplanning and Physical Synthesis with First Encounter

Chip implementation

Low-Power Implementation

Nanometer routing

Detail Routing with NanoRoute Router

Extraction, and sign-off power analysis

Cadence QRC Extraction

VoltageStorm Power Rail Analysis

Basic Static Timing Analysis

Encounter Timing System

Timing and signal integrity analysis

Encounter Conformal Constraint Designer

Logic Equivalence Checking with Encounter Conformal EC

Advanced Logic Equivalence Checking with Encounter Conformal EC

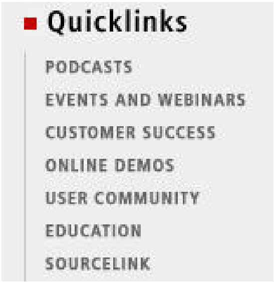
Equivalence checking

Course names

Topics covered In the courses

For more information about Cadence® courses:

- Go to **www.cadence.com**.
- Click **Education** in the *Quicklinks* section.



Quicklinks

PODCASTS

EVENTS AND WEBINARS

CUSTOMER SUCCESS

ONLINE DEMOS

USER COMMUNITY

EDUCATION

SOURCELINK

- Click the **Course catalog** link near the top of the center column.
- Click a Cadence technology platform (such as **Digital IC Design**).
- Click a course name.

The browser displays a course description and gives you an opportunity to enroll.

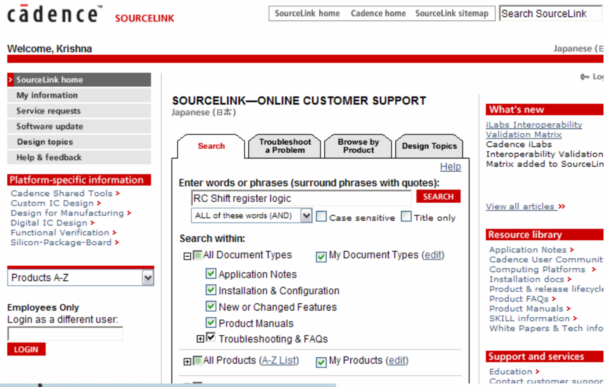
Getting Help with Encounter RTL Compiler

Self Help

- Click the **Help** button in the upper right corner of the Encounter window.
- Use SourceLink online support: http://sourcelink.cadence.com
  You need a subscription to access this support.

Cadence Users Group

- Use http://www.cdnusers.org
  Select **Digital IC** to view solutions and recommendations from other Cadence users on tools and methodologies.



Welcome, Krishna

SOURCELINK—ONLINE CUSTOMER SUPPORT
Japanese (日本)

Enter words or phrases (surround phrases with quotes): RC Shift register logic

ALL of these words (AND) □ Case sensitive □ Title only

Search within:

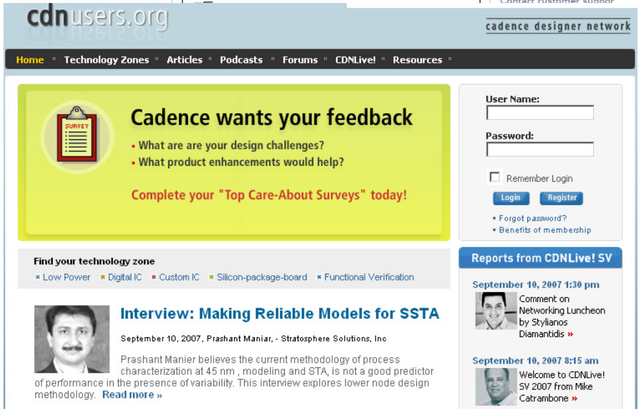
⊟ All Document Types ☑ My Document Types (edit)

☑ Application Notes
☑ Installation & Configuration
☑ New or Changed Features
☑ Product Manuals
⊞ ☑ Troubleshooting & FAQs

⊞ All Products (A-Z List) ☑ My Products (edit)

iLabs Interoperability Validation Matrix
Cadence iLabs Interoperability Validation Matrix added to SourceLin

Employees Only
Login as a different user:



Cadence wants your feedback
- What are are your design challenges?
- What product enhancements would help?

Complete your "Top Care-About Surveys" today!

User Name:
Password:
□ Remember Login

Find your technology zone
Low Power · Digital IC · Custom IC · Silicon-package-board · Functional Verification

Interview: Making Reliable Models for SSTA

September 10, 2007, Prashant Maniar, - Stratosphere Solutions, Inc

Prashant Manier believes the current methodology of process characterization at 45 nm , modeling and STA, is not a good predictor of performance in the presence of variability. This interview explores lower node design methodology. Read more »

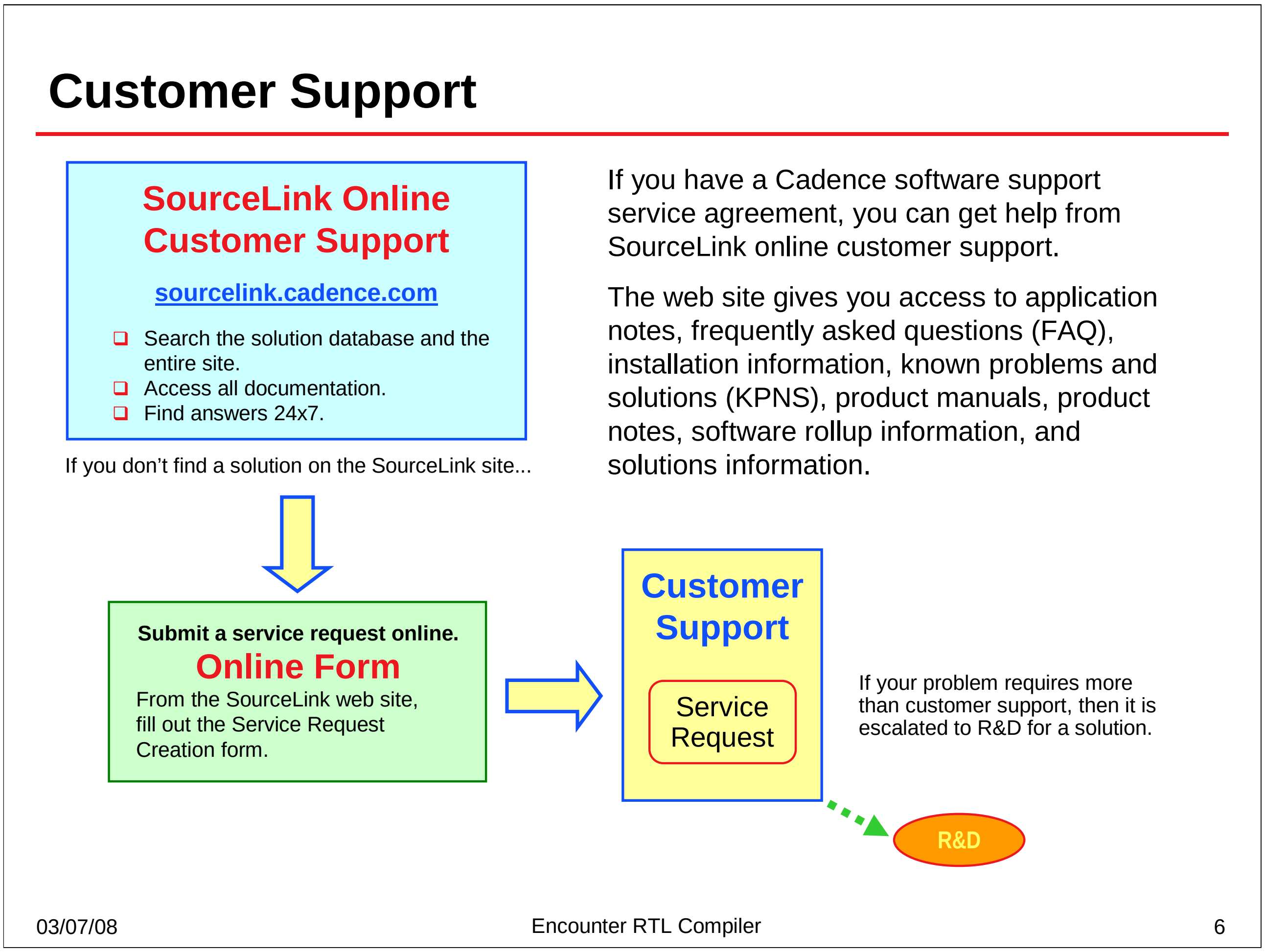
September 10, 2007 1:30 pm
Comment on Networking Luncheon by Stylianos Diamantidis »

September 10, 2007 8:15 am
Welcome to CDNLive! SV 2007 from Mike Catrambone »

The **Help** button in the software brings up the software documentation. You can also access the SourceLink® online customer support from the Help button in the main software window.

The SourceLink online support site provides not only the latest manuals but gives you access to many other features. It lets you retrieve the latest software updates, submit service requests, view solutions, and access FAQs.

The Cadence Users Group site provides still more information from other users like you.



Customer Support

SourceLink Online Customer Support

sourcelink.cadence.com

- ❑ Search the solution database and the entire site.
- ❑ Access all documentation.
- ❑ Find answers 24x7.

If you don’t find a solution on the SourceLink site...

Submit a service request online.

Online Form

From the SourceLink web site, fill out the Service Request Creation form.

Customer Support

Service Request

If your problem requires more than customer support, then it is escalated to R&D for a solution.

R&D

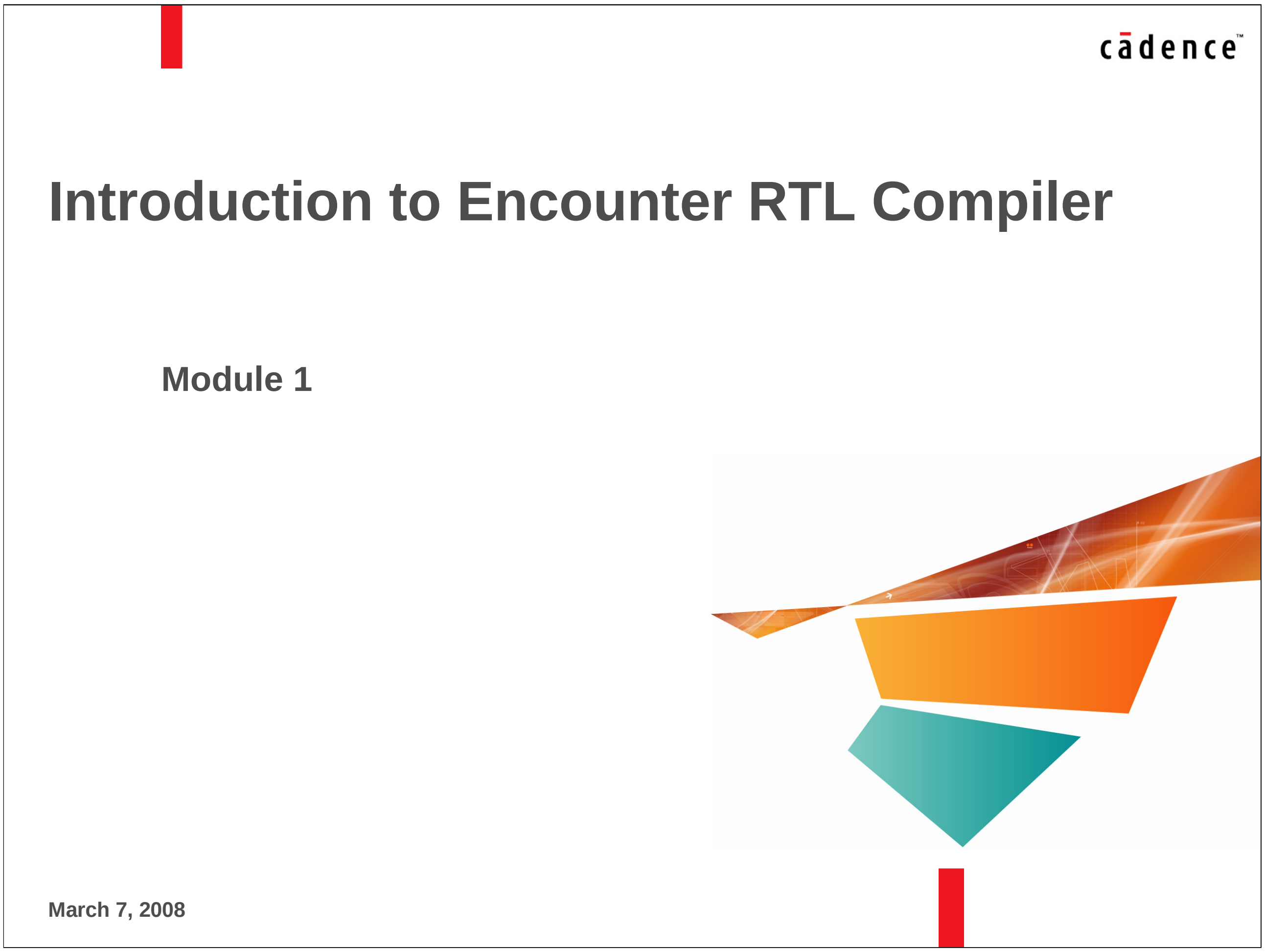
If you have a Cadence software support service agreement, you can get help from SourceLink online customer support.

The web site gives you access to application notes, frequently asked questions (FAQ), installation information, known problems and solutions (KPNS), product manuals, product notes, software rollup information, and solutions information.

To view information in SourceLink online support:

1. Point your web browser to sourcelink.cadence.com.
2. Log in.
3. Enter search criteria.

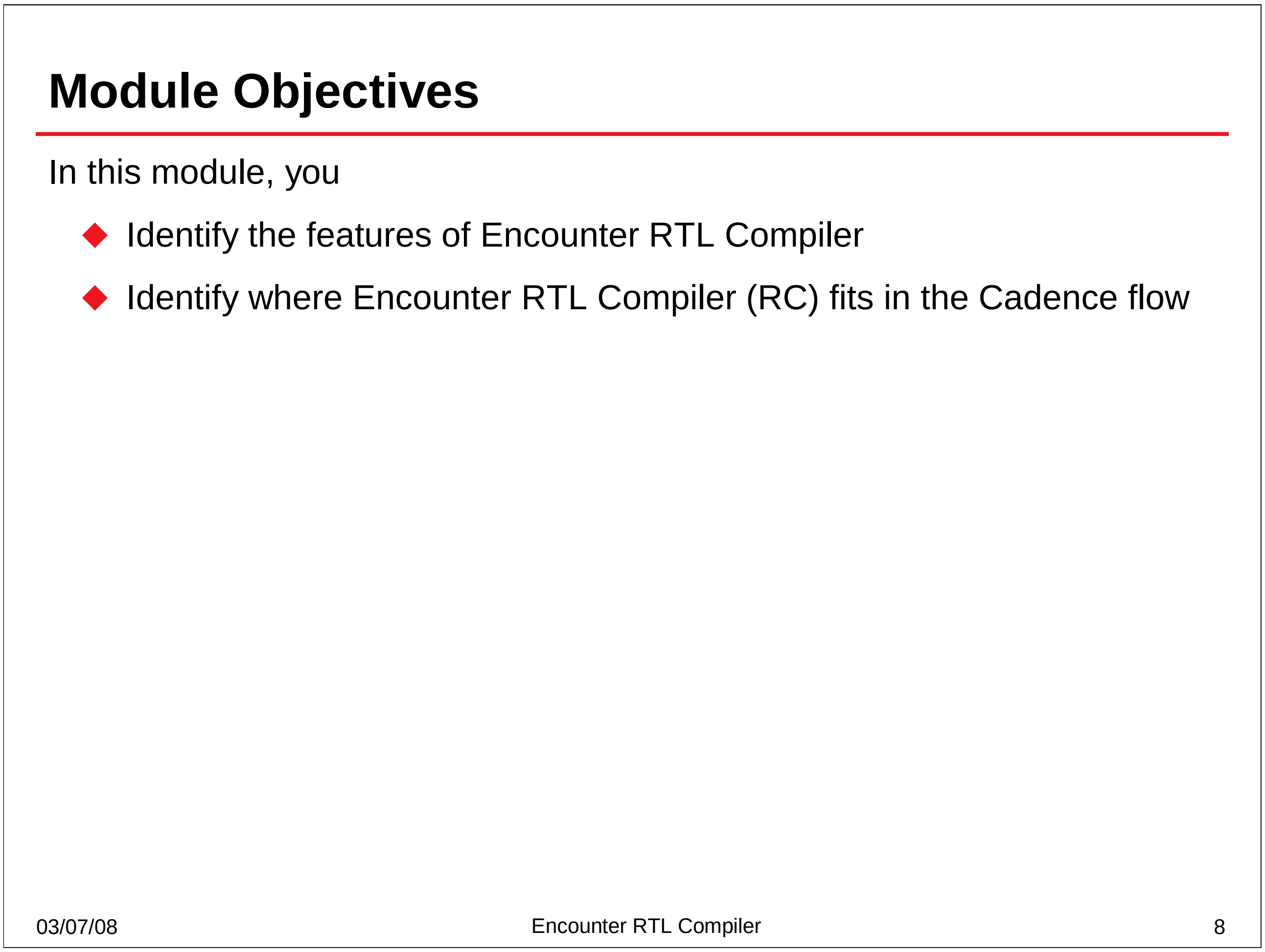
You can search by product, release, document type, or keyword. You can also browse by product, release, or document type.



Introduction to Encounter RTL Compiler

Module 1

March 7, 2008

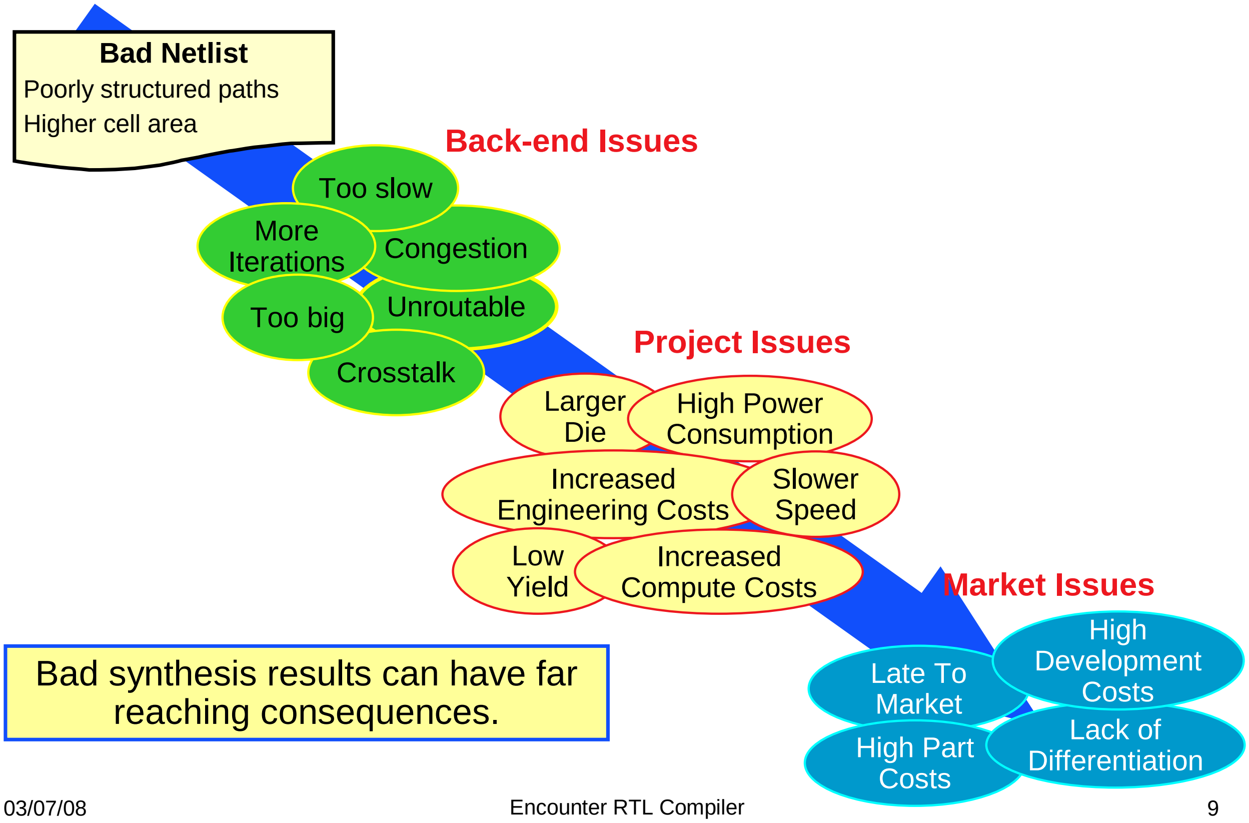


Module Objectives

In this module, you

- Identify the features of Encounter RTL Compiler
- Identify where Encounter RTL Compiler (RC) fits in the Cadence flow

Does RTL Synthesis Really Matter?



Bad Netlist

Poorly structured paths

Higher cell area

Back-end Issues

Too slow

More Iterations

Congestion

Too big

Unroutable

Crosstalk

Project Issues

Larger Die

High Power Consumption

Increased Engineering Costs

Slower Speed

Low Yield

Increased Compute Costs

Market Issues

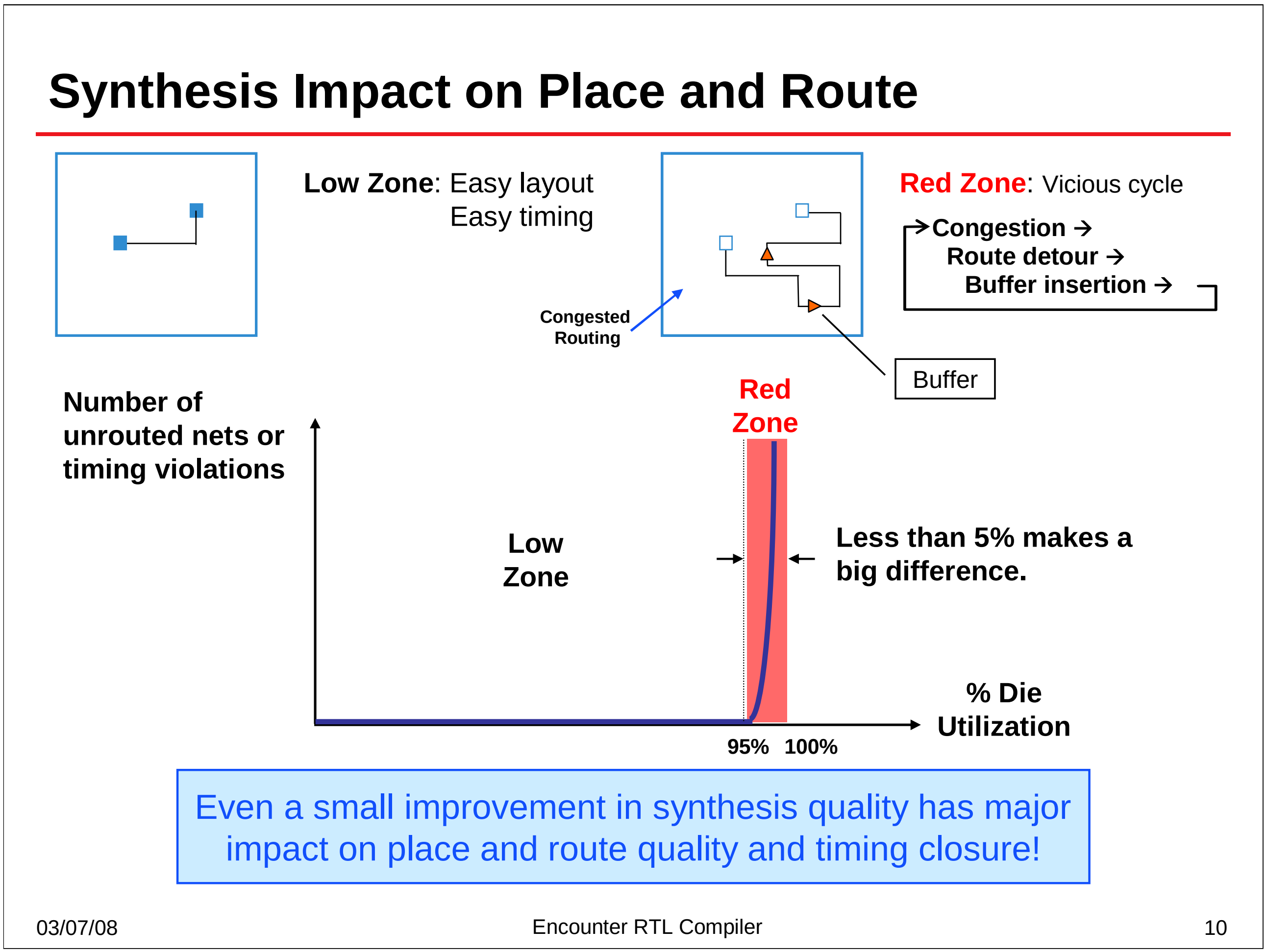
Late To Market

High Development Costs

High Part Costs

Lack of Differentiation

Bad synthesis results can have far reaching consequences.

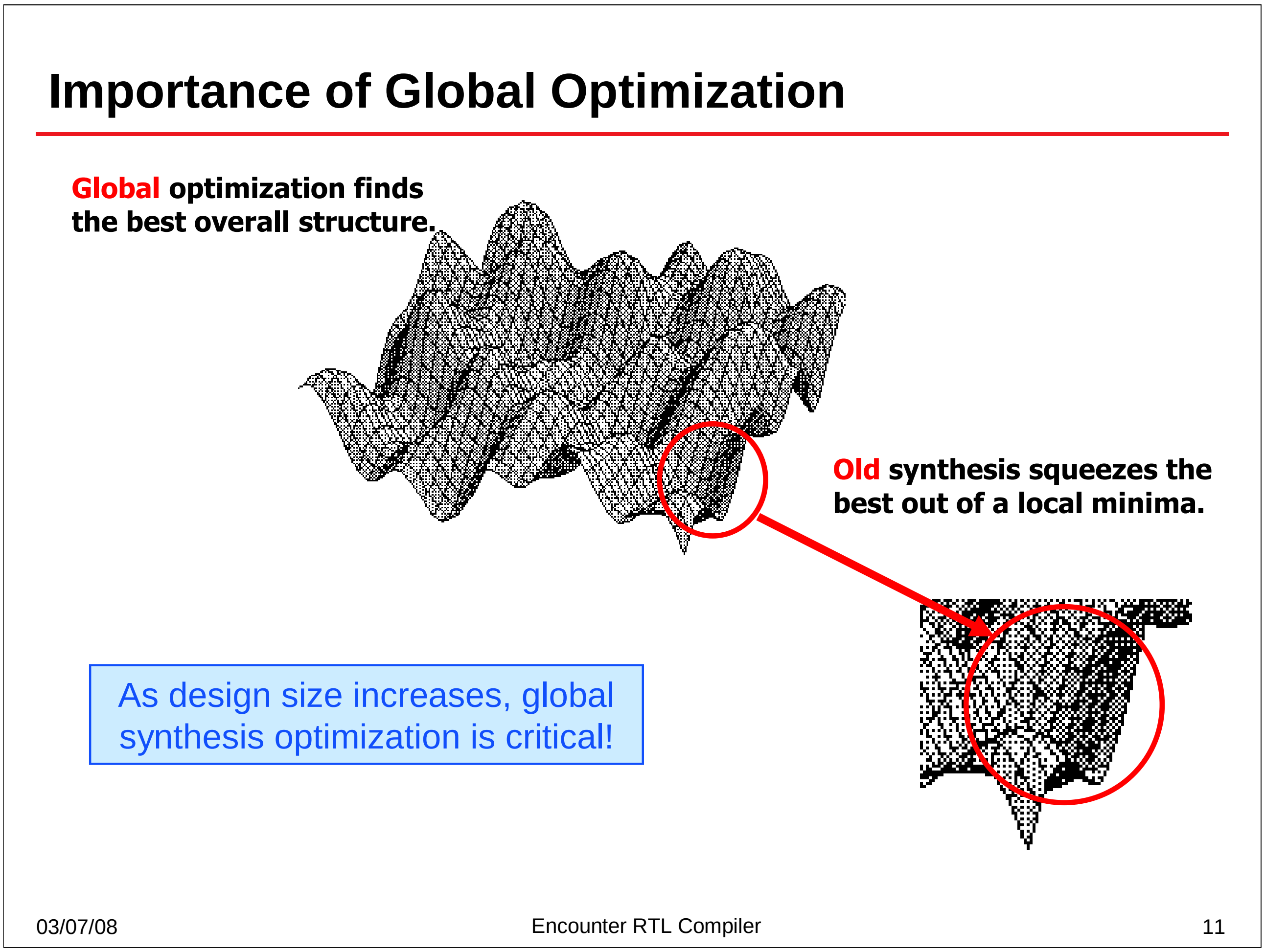


Synthesis Impact on Place and Route

Even a small improvement in synthesis quality has major impact on place and route quality and timing closure!

As the die utilization approaches the red zone (95% to 100%), meeting timing becomes increasingly difficult. As utilization and cell density increase, more congestion in interconnect appears. The result is a vicious cycle of detoured routes, which result in longer wires. Long wires require the insertion of buffers. Buffering increases the cell density, which increases congestion, and so on.

Thus, for a given die size, 3% to 5% reduction in cell area helps the design to be congestion free and provides enough resources to route.



Importance of Global Optimization

Global optimization finds the best overall structure.

Old synthesis squeezes the best out of a local minima.

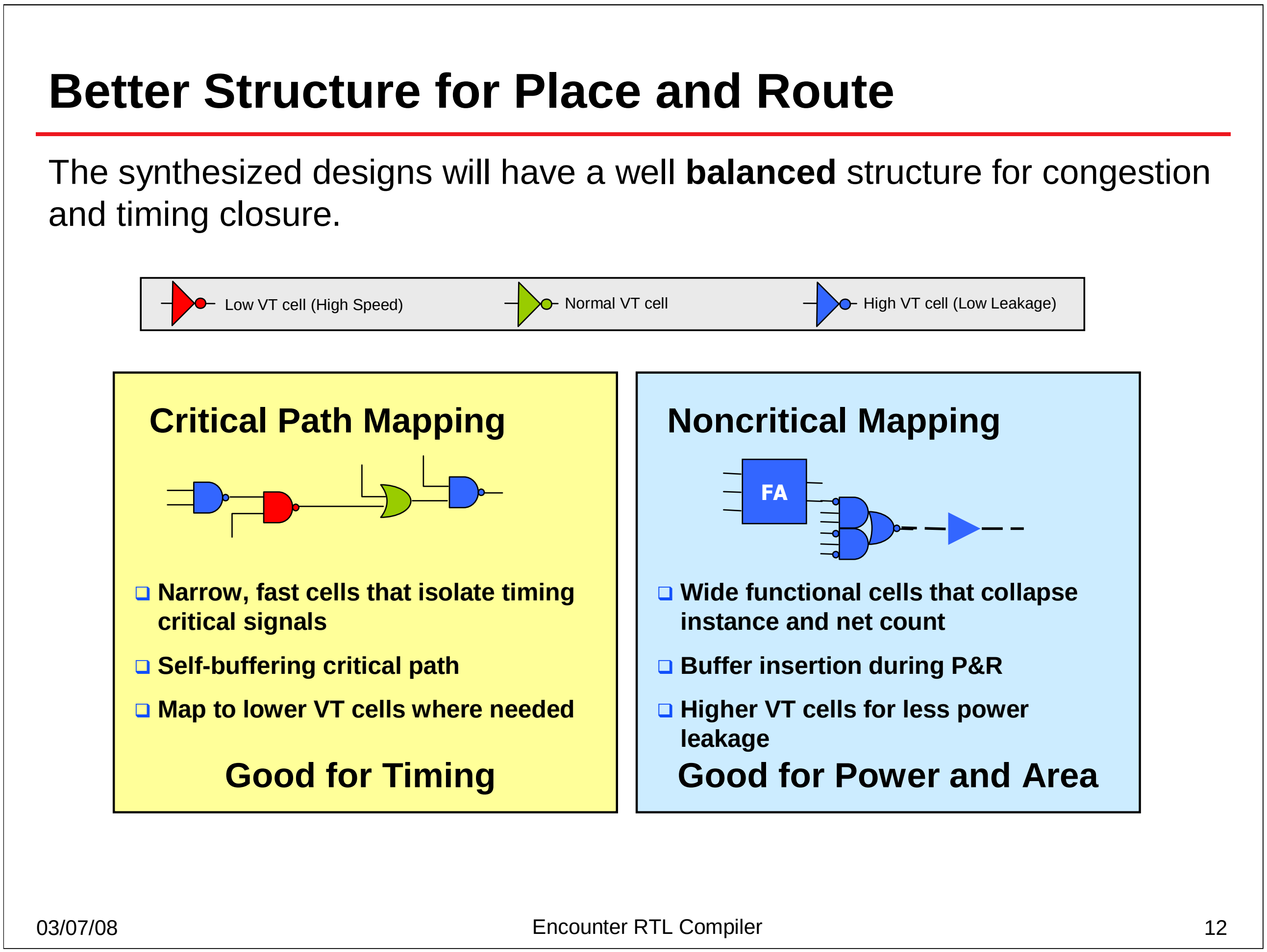
As design size increases, global synthesis optimization is critical!

The old synthesis tools work at incrementally refining the optimization cost function (represented by the 3-D graph) providing the best results from an initial starting point (the initial logic structure), but result in a poor job of recovering from a bad starting point.

In contrast, Encounter® RTL Compiler algorithms iterate through multiple design scenarios that provide the best initial design to create a solution that is far superior to those achievable using the other synthesis tools.

For example, in the RTL, a MUX drives the inputs of a multiplier. The old synthesis tools might not find this MUX redundant and might use the delay of the MUX in the final slack numbers. If it is redundant logic, this MUX could have been eliminated prior to synthesis. Global synthesis determines the best initial generic netlist to use as a starting point.

As designs get larger, the benefits of a global optimization strategy become increasingly important.


Low VT cell (High Speed)
Normal VT cell
High VT cell (Low Leakage)
FA

Better Structure for Place and Route

The synthesized designs will have a well **balanced** structure for congestion and timing closure.

Critical Path Mapping

- Narrow, fast cells that isolate timing critical signals
- Self-buffering critical path
- Map to lower VT cells where needed

Good for Timing

Noncritical Mapping

- Wide functional cells that collapse instance and net count
- Buffer insertion during P&R
- Higher VT cells for less power leakage

Good for Power and Area

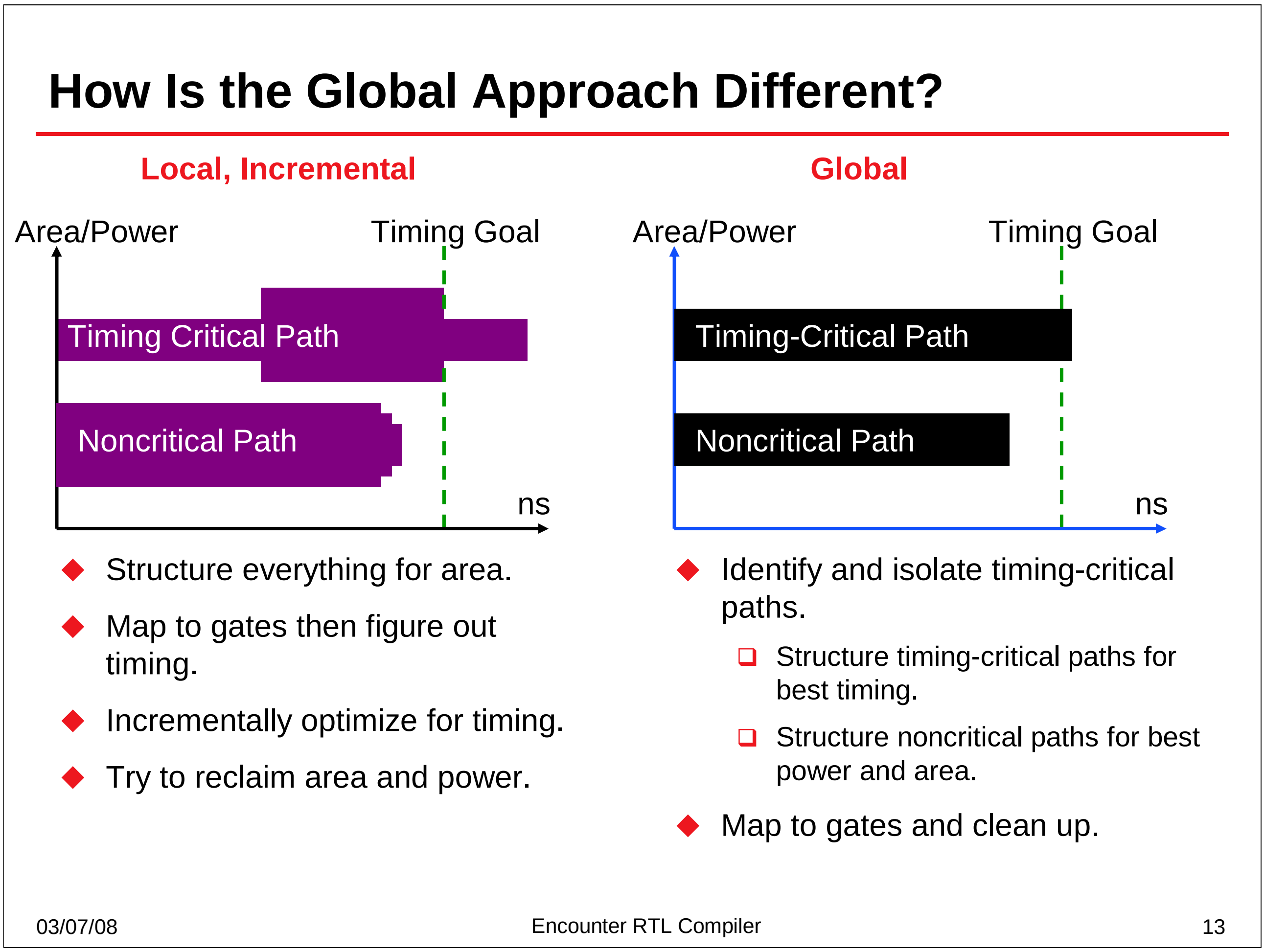
03/07/08 Encounter RTL Compiler 12

Global synthesis, in its two-pass sweep of the design and libraries, identifies critical and noncritical paths before structuring. Thus, the compiler can better isolate critical paths from noncritical, and structure them differently.

Smaller cells have a richer set of sizing options, including multi-VT cells. Critical paths are structured to be faster using smaller cells. This strategy creates self-buffering paths that will be helpful to the downstream placement tools.

Noncritical paths are structured to be smaller and to cause less leakage. Encounter RTL Compiler will optimize for leakage even with a single VT library.

The compiler also reduces the instance count and net count for the noncritical paths to improve the optimization of place and route.



How Is the Global Approach Different?

Local, Incremental

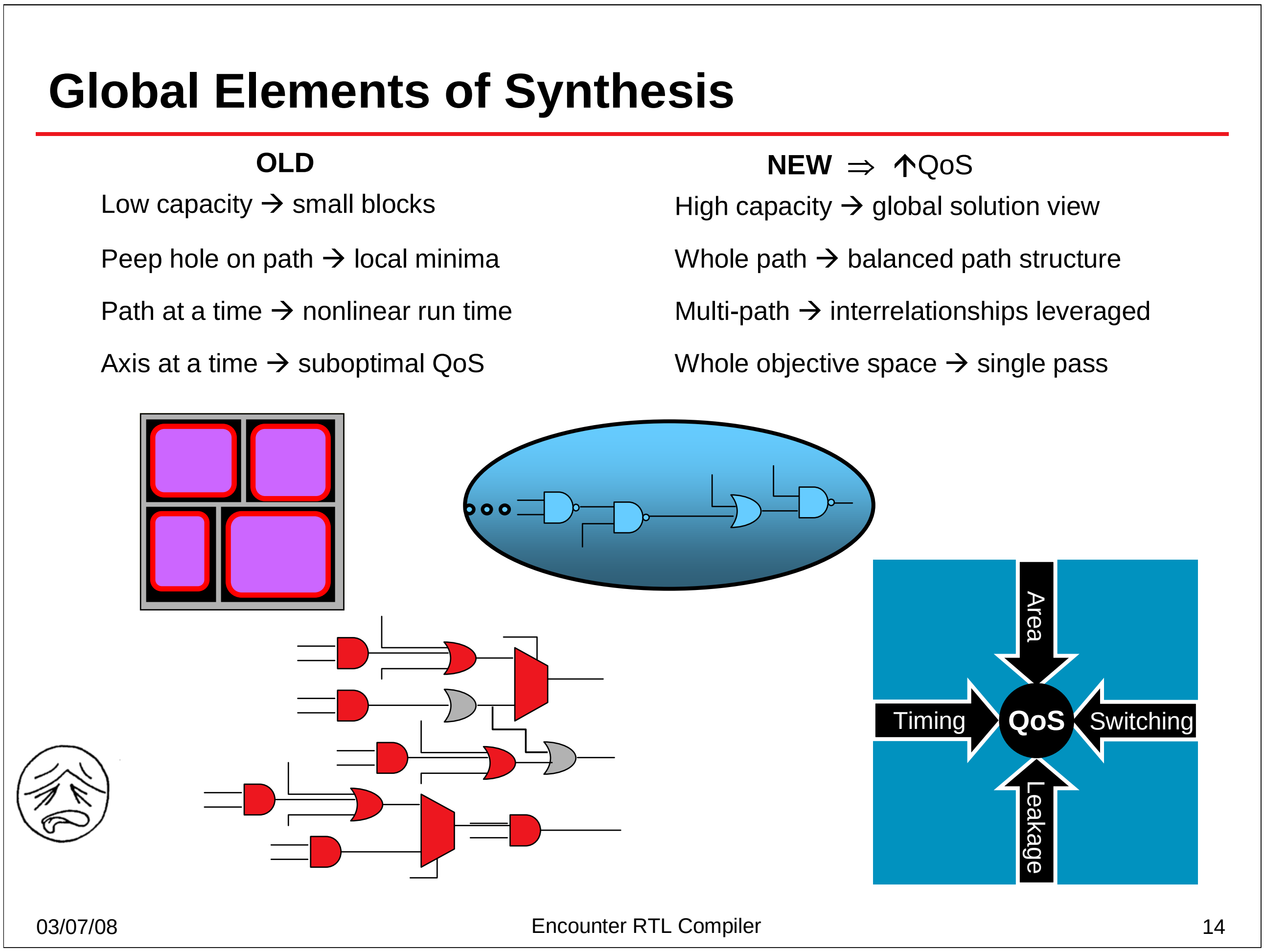
- Structure everything for area.
- Map to gates then figure out timing.
- Incrementally optimize for timing.
- Try to reclaim area and power.

Global

- Identify and isolate timing-critical paths.
  - Structure timing-critical paths for best timing.
  - Structure noncritical paths for best power and area.
- Map to gates and clean up.

This graphic compares how the local and global approaches translate to path structures in a design.

Note that the length of the horizontal bars represent timing path length and the width represents area/power consumed by that path.

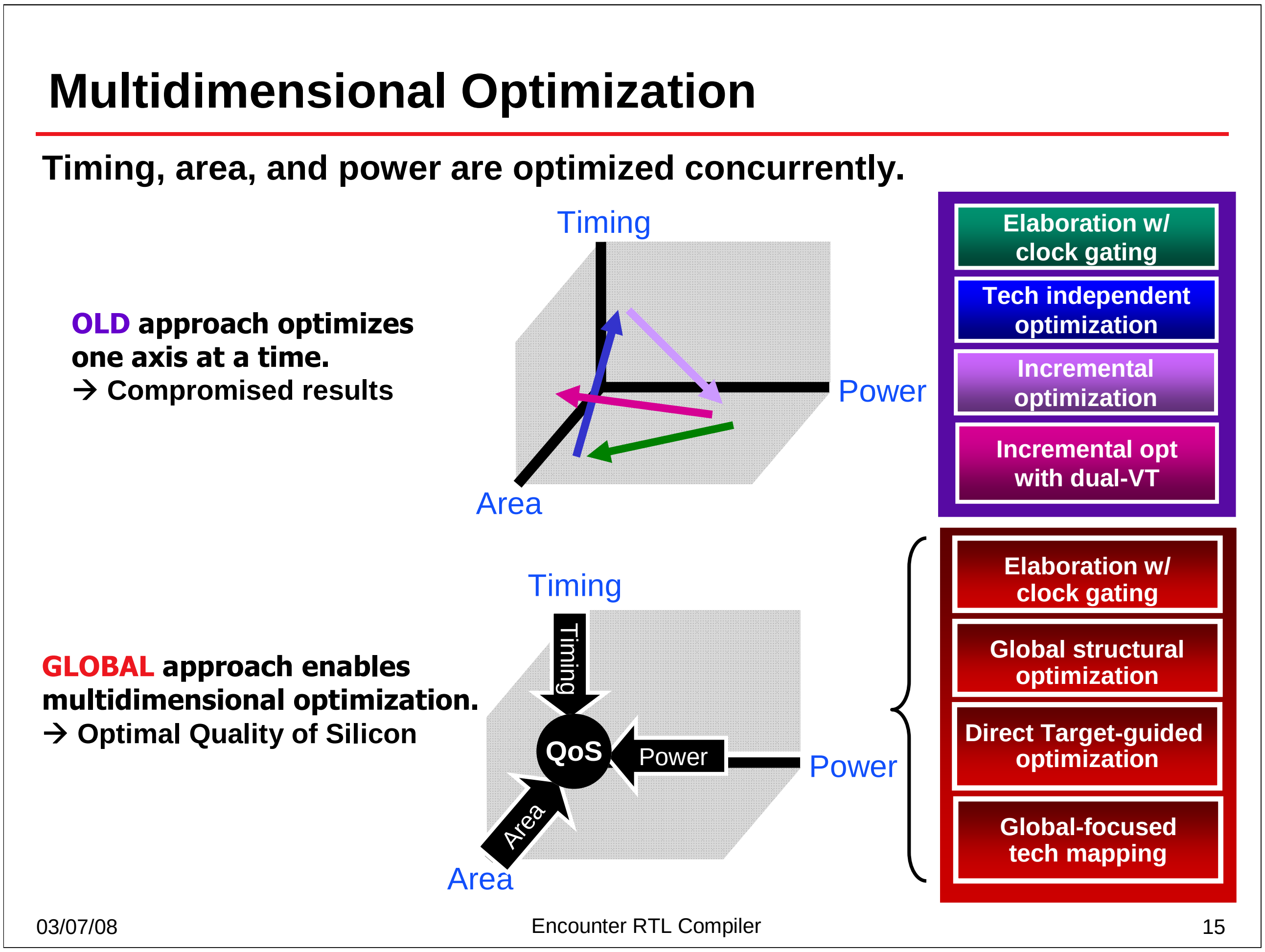


Global Elements of Synthesis

OLD	NEW ⇒ ↑QoS
Low capacity → small blocks	High capacity → global solution view
Peep hole on path → local minima	Whole path → balanced path structure
Path at a time → nonlinear run time	Multi-path → interrelationships leveraged
Axis at a time → suboptimal QoS	Whole objective space → single pass

03/07/08 Encounter RTL Compiler 14

QoS: Quality of Silicon



Multidimensional Optimization

Timing, area, and power are optimized concurrently.

OLD approach optimizes one axis at a time.
→ Compromised results

Elaboration w/ clock gating

Tech independent optimization

Incremental optimization

Incremental opt with dual-VT

GLOBAL approach enables multidimensional optimization.
→ Optimal Quality of Silicon

Elaboration w/ clock gating

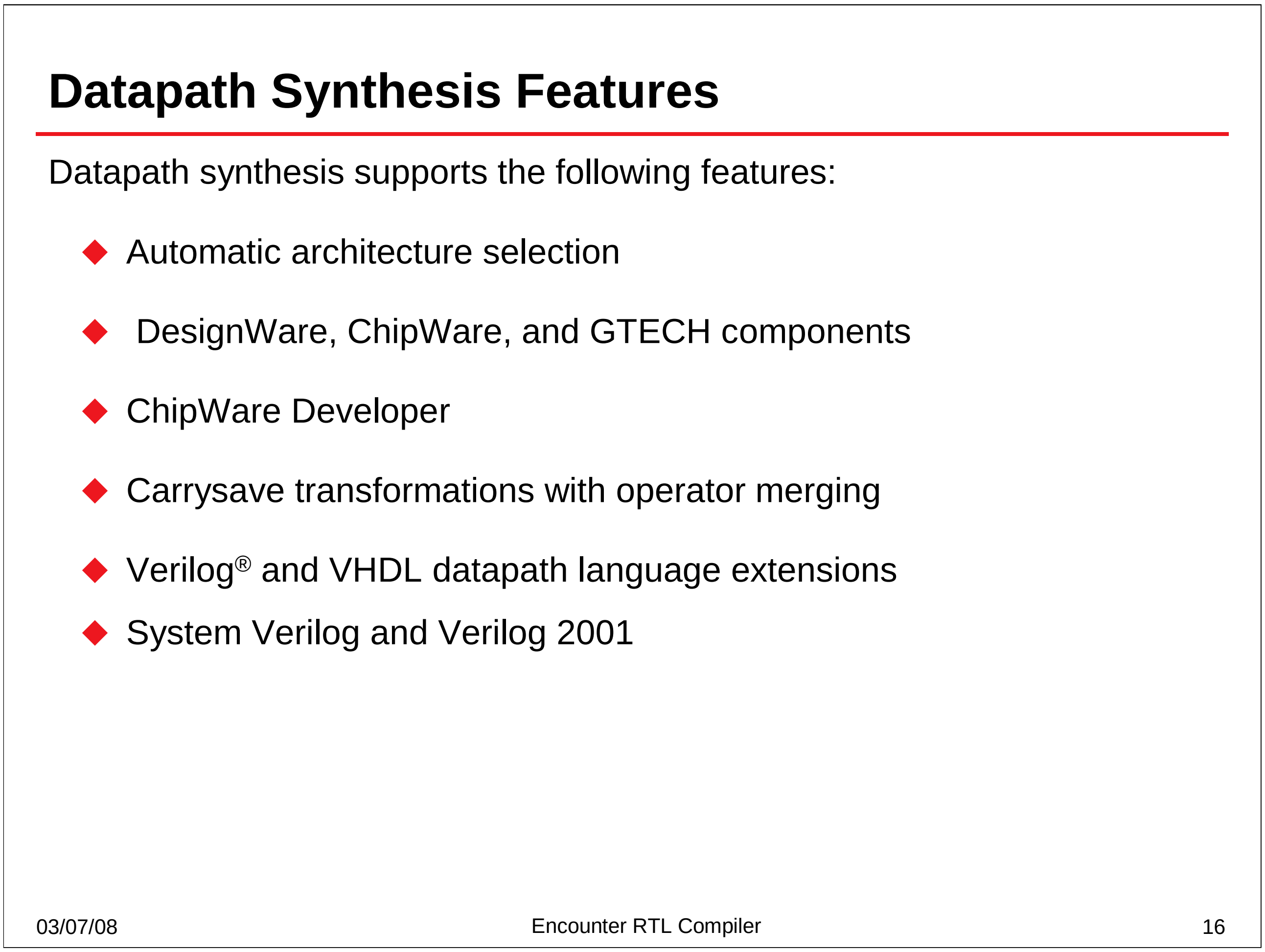
Global structural optimization

Direct Target-guided optimization

Global-focused tech mapping

03/07/08 Encounter RTL Compiler 15

Synthesis based on the old algorithms can only optimize for one objective at a time. This limitation results in iterations and eventually long run times and compromised results. RTL Compiler global optimization optimizes for timing while using power and area as cost factors. This optimization results in a single pass flow that creates the desired balance of timing, power, and area.

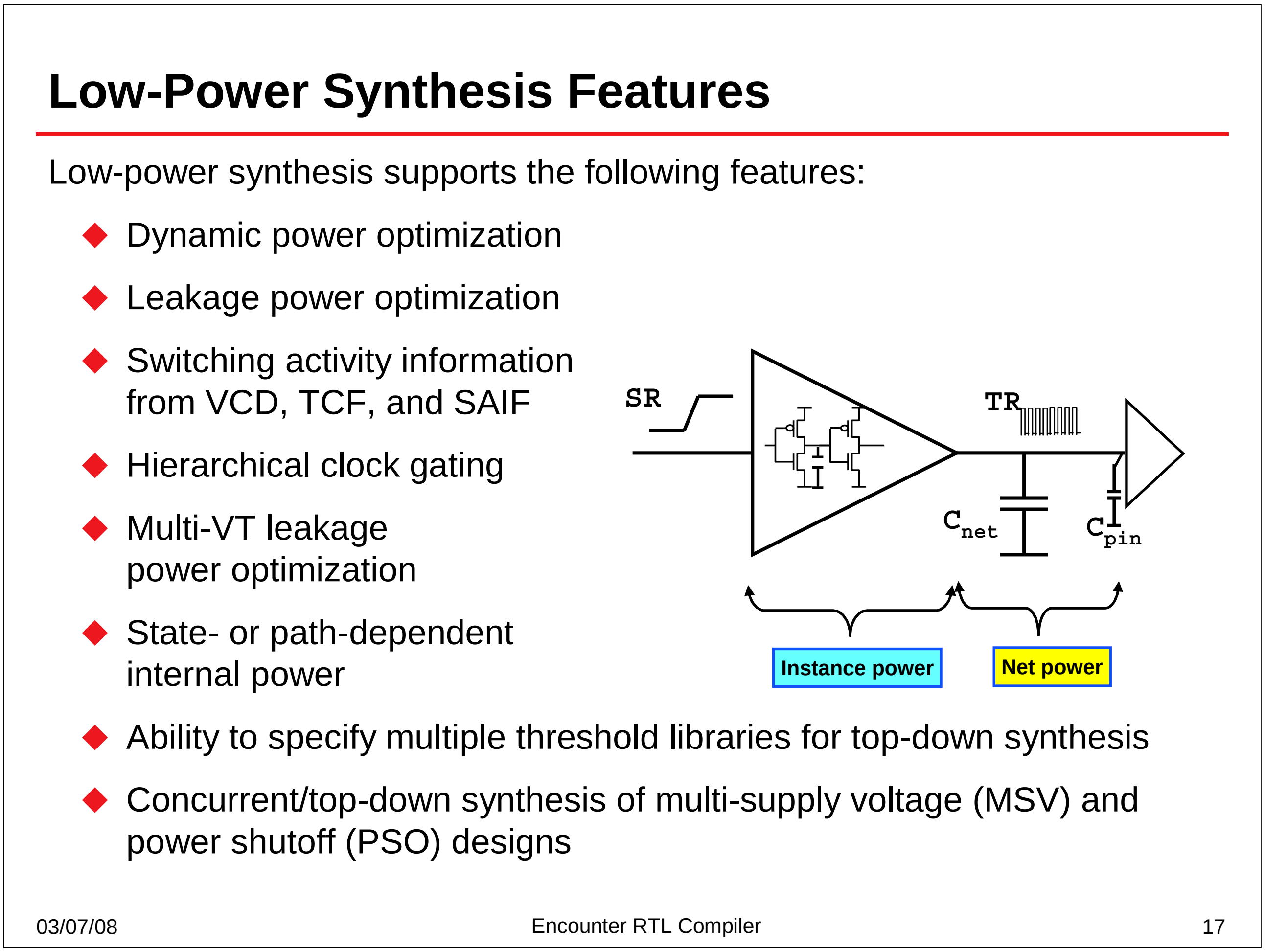


Datapath Synthesis Features

Datapath synthesis supports the following features:

- Automatic architecture selection
- DesignWare, ChipWare, and GTECH components
- ChipWare Developer
- Carrysave transformations with operator merging
- Verilog® and VHDL datapath language extensions
- System Verilog and Verilog 2001

03/07/08 Encounter RTL Compiler 16

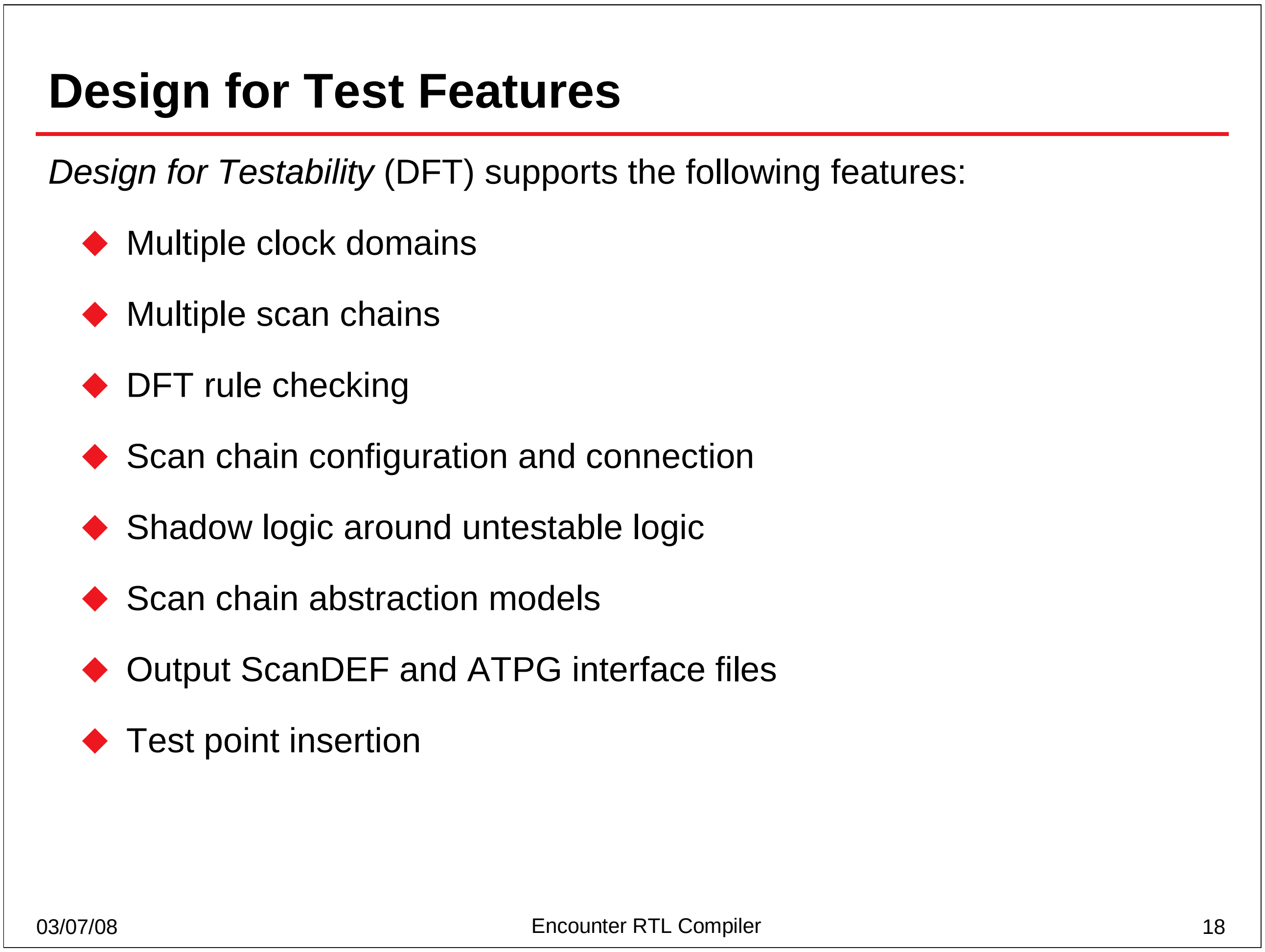


Low-Power Synthesis Features

Low-power synthesis supports the following features:

- Dynamic power optimization
- Leakage power optimization
- Switching activity information from VCD, TCF, and SAIF
- Hierarchical clock gating
- Multi-VT leakage power optimization
- State- or path-dependent internal power
- Ability to specify multiple threshold libraries for top-down synthesis
- Concurrent/top-down synthesis of multi-supply voltage (MSV) and power shutoff (PSO) designs

TCF:	Toggle count format
SAIF:	Switching activity information format
VCD:	Value change dump file
MSV:	Multiple supply voltage
PSO:	Power Shutoff
VT:	Voltage Threshold (V_{th})

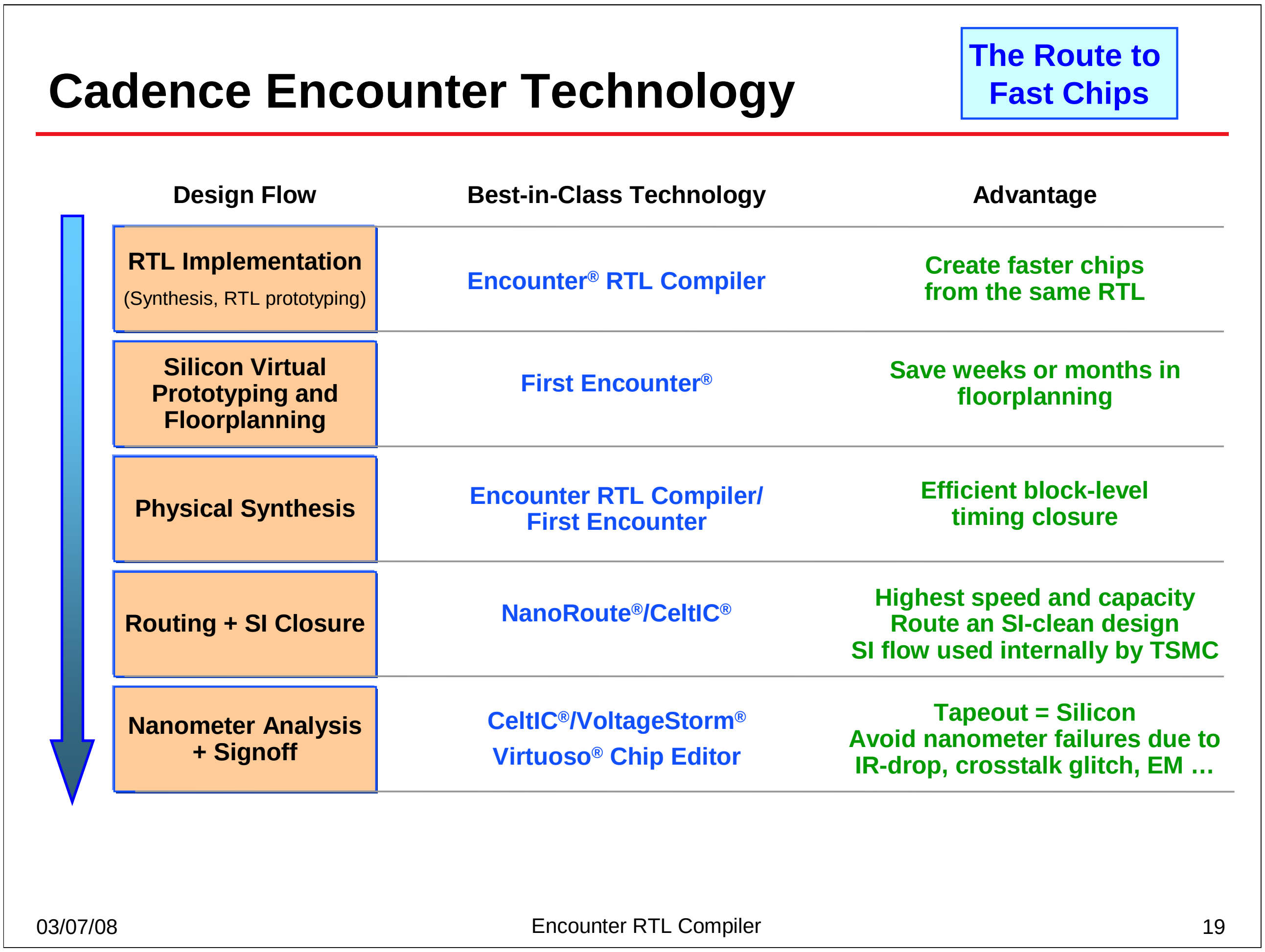


Design for Test Features

Design for Testability (DFT) supports the following features:

- Multiple clock domains
- Multiple scan chains
- DFT rule checking
- Scan chain configuration and connection
- Shadow logic around untestable logic
- Scan chain abstraction models
- Output ScanDEF and ATPG interface files
- Test point insertion

ATPG – Automatic Test Pattern Generator



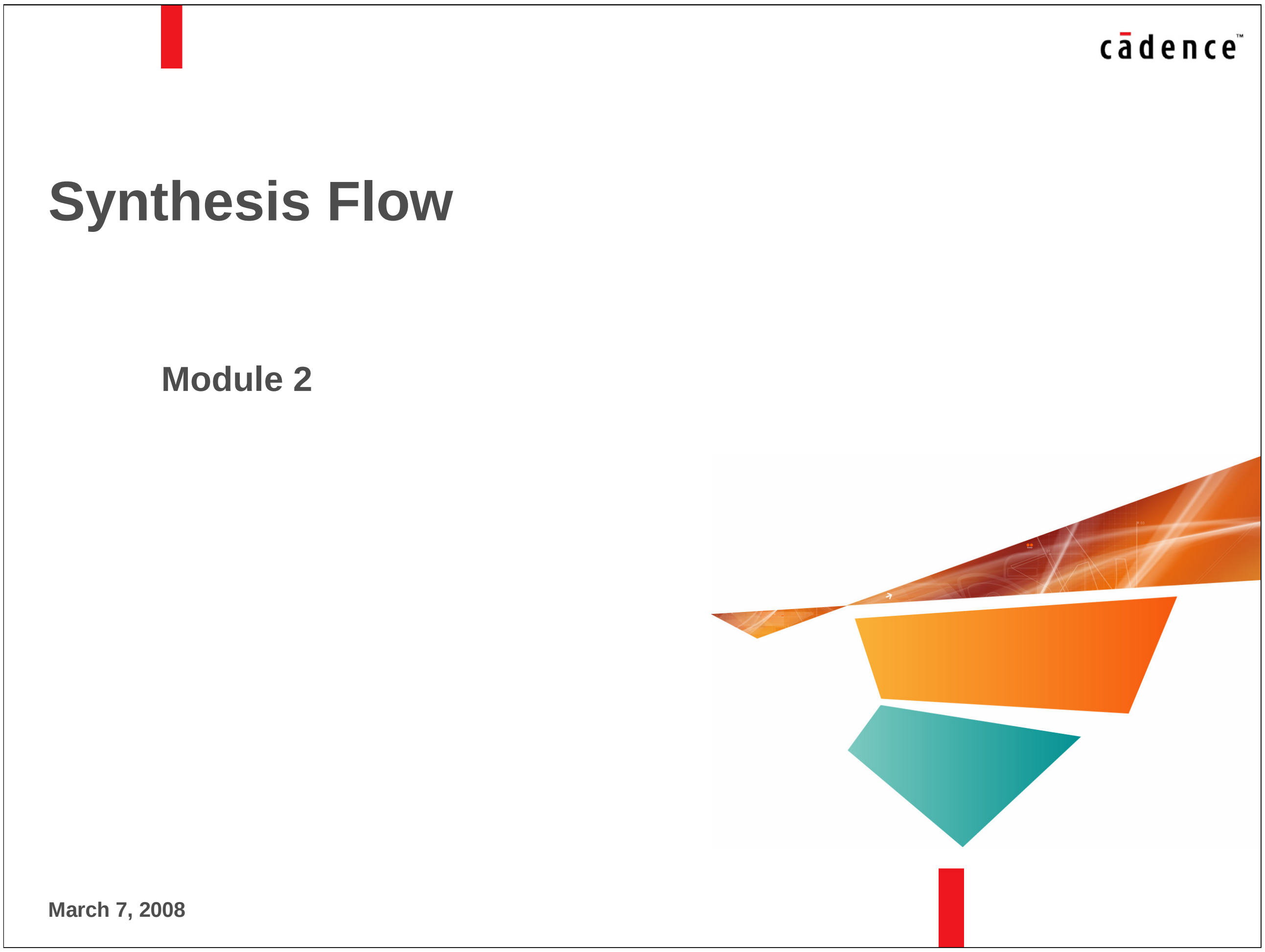
Cadence Encounter Technology

The Route to Fast Chips

Design Flow	Best-in-Class Technology	Advantage
RTL Implementation (Synthesis, RTL prototyping)	Encounter® RTL Compiler	Create faster chips from the same RTL
Silicon Virtual Prototyping and Floorplanning	First Encounter®	Save weeks or months in floorplanning
Physical Synthesis	Encounter RTL Compiler/ First Encounter	Efficient block-level timing closure
Routing + SI Closure	NanoRoute®/CeltIC®	Highest speed and capacity Route an SI-clean design SI flow used internally by TSMC
Nanometer Analysis + Signoff	CeltIC®/VoltageStorm® Virtuoso® Chip Editor	Tapeout = Silicon Avoid nanometer failures due to IR-drop, crosstalk glitch, EM …

03/07/08 Encounter RTL Compiler 19

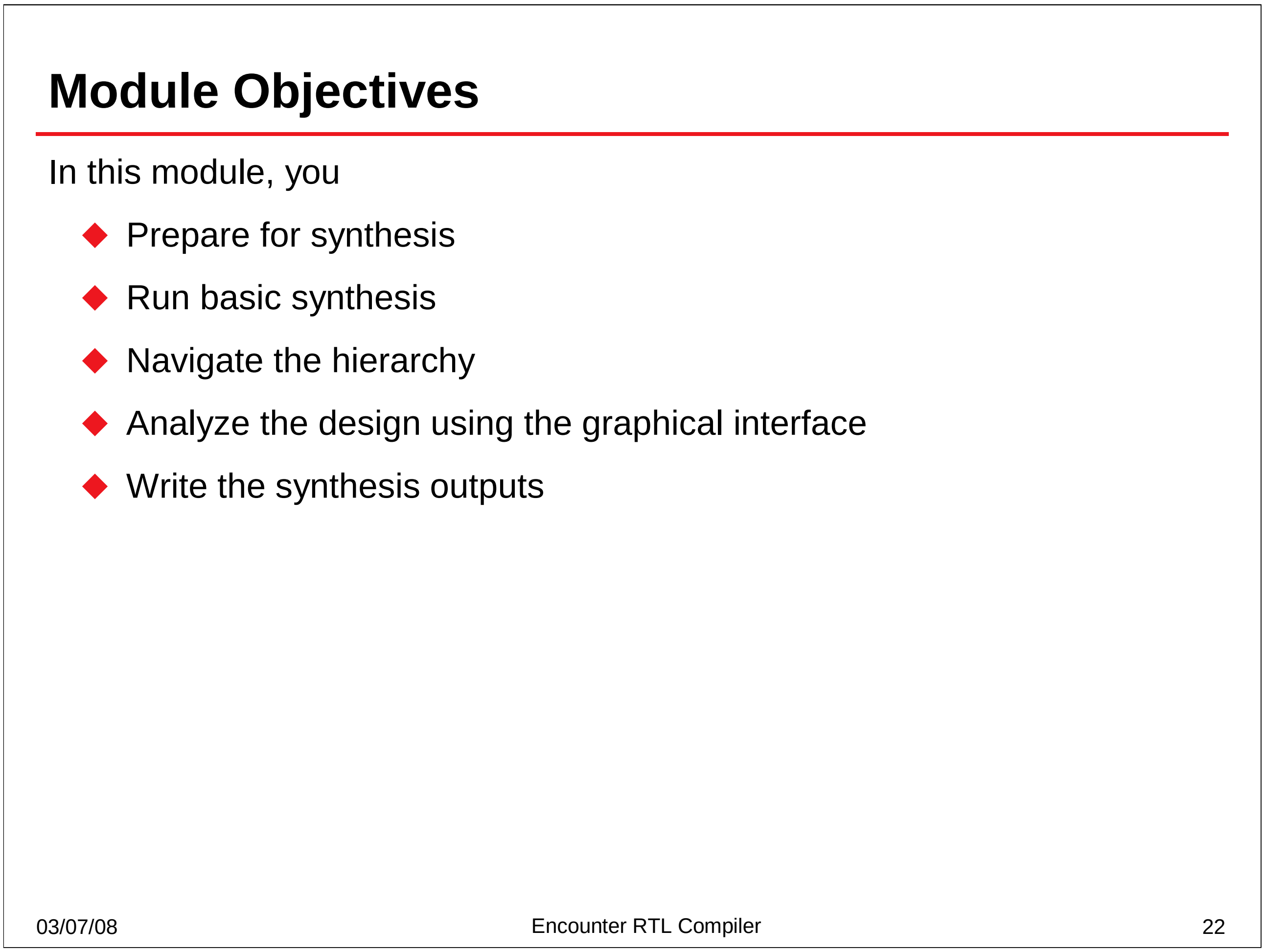
EM stands for electromigration.



Synthesis Flow

Module 2

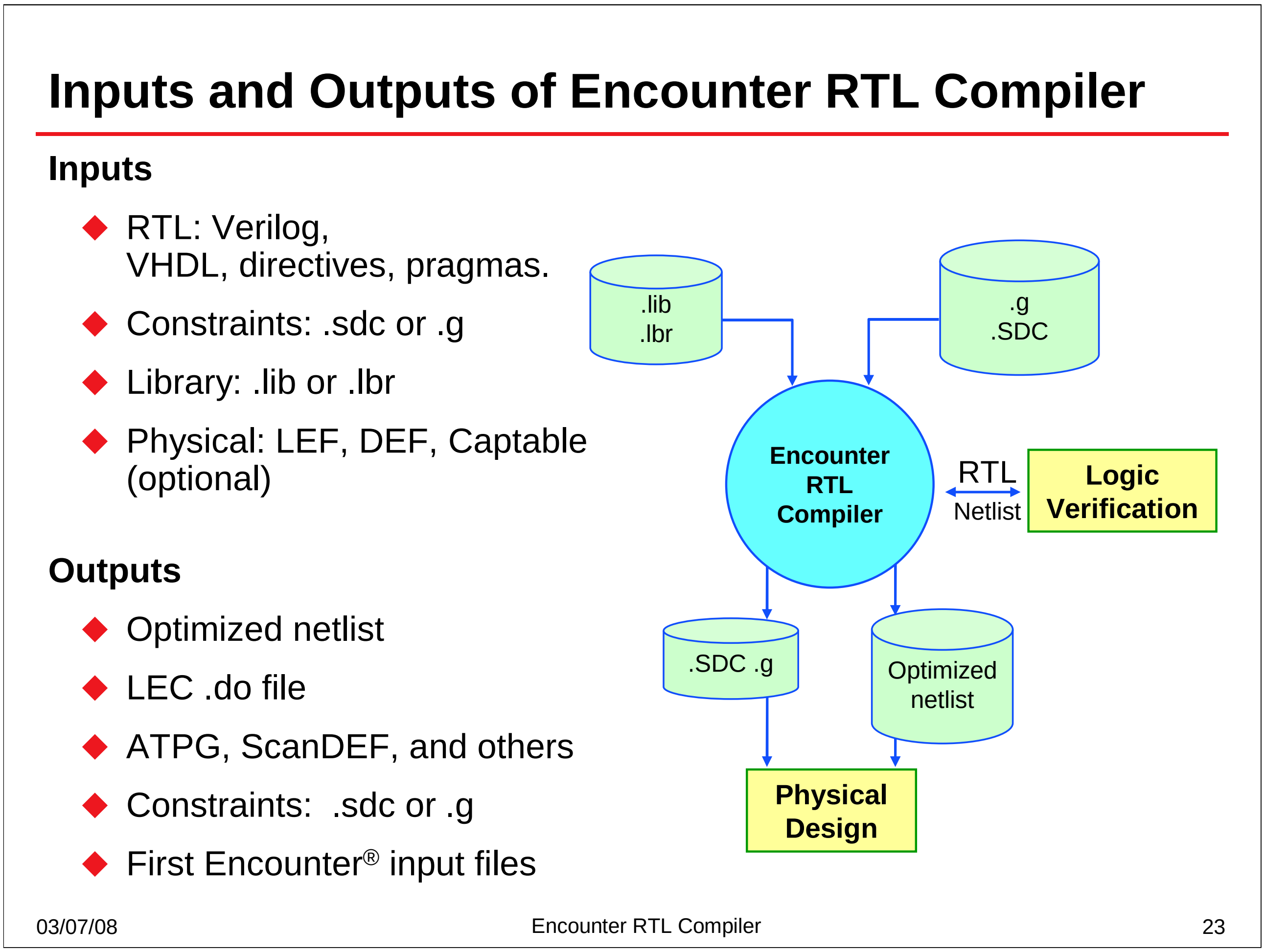
March 7, 2008



Module Objectives

In this module, you

- Prepare for synthesis
- Run basic synthesis
- Navigate the hierarchy
- Analyze the design using the graphical interface
- Write the synthesis outputs



Inputs and Outputs of Encounter RTL Compiler

Inputs

- RTL: Verilog, VHDL, directives, pragmas.
- Constraints: .sdc or .g
- Library: .lib or .lbr
- Physical: LEF, DEF, Captable (optional)

Outputs

- Optimized netlist
- LEC .do file
- ATPG, ScanDEF, and others
- Constraints: .sdc or .g
- First Encounter® input files

ATPG – Automatic Test Pattern Generation

DEF – Design Exchange Format

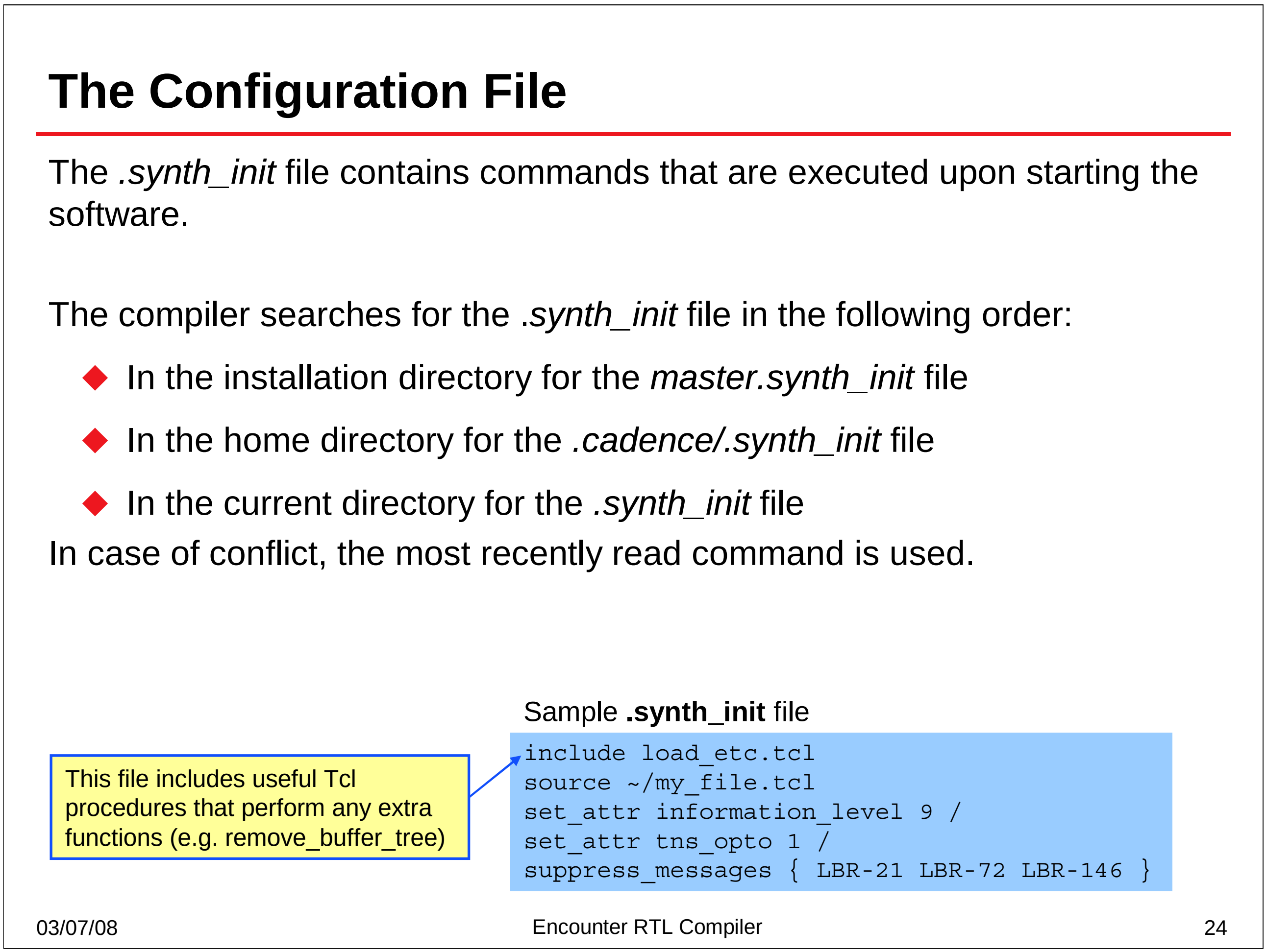
.G – Encounter® RTL Compiler design constraints

LEF – Library Exchange Format

.LBR – Encounter RTL Compiler library (binary format)

.LIB – Synopsys library format

.SDC – Synopsys design constraints



The Configuration File

The *.synth_init* file contains commands that are executed upon starting the software.

The compiler searches for the *.synth_init* file in the following order:

- In the installation directory for the *master.synth_init* file
- In the home directory for the *.cadence/.synth_init* file
- In the current directory for the *.synth_init* file

In case of conflict, the most recently read command is used.

This file includes useful Tcl procedures that perform any extra functions (e.g. remove_buffer_tree)

Sample **.synth_init** file

```
include load_etc.tcl
source ~/my_file.tcl
set_attr information_level 9 /
set_attr tns_opto 1 /
suppress_messages { LBR-21 LBR-72 LBR-146 }
```

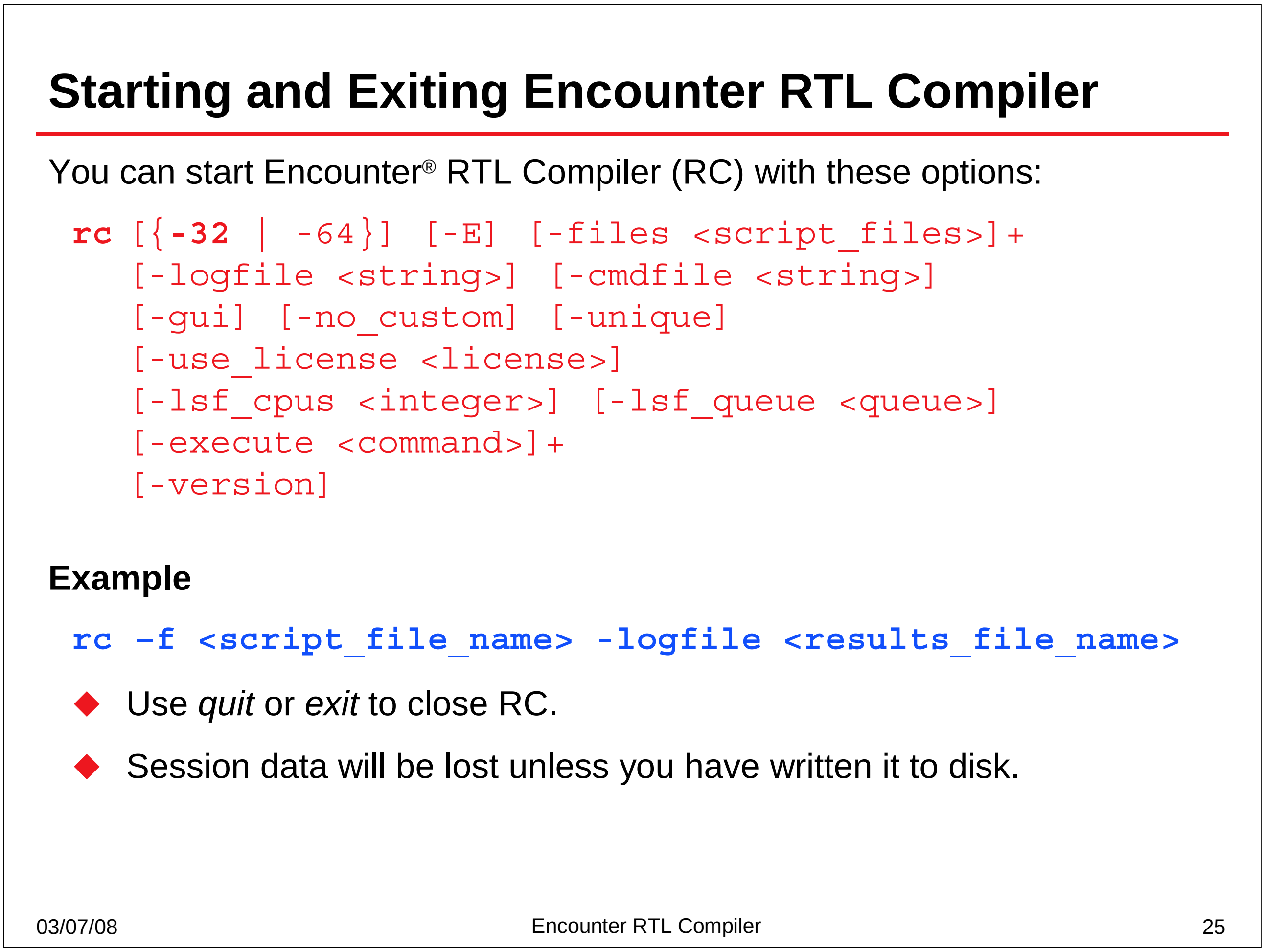
In the example shown above, the *.synth_init* file does the following:

Load some Tcl procedures

Suppress unwanted warning or information messages

Set globally applicable attributes

The compiler automatically loads these files unless you start it with the *-no_custom* option, in which case the compiler reads only the configuration file from the installation directory.



Starting and Exiting Encounter RTL Compiler

You can start Encounter® RTL Compiler (RC) with these options:

```
rc [{-32 | -64}] [-E] [-files <script_files>]+
   [-logfile <string>] [-cmdfile <string>]
   [-gui] [-no_custom] [-unique]
   [-use_license <license>]
   [-lsf_cpus <integer>] [-lsf_queue <queue>]
   [-execute <command>]+
   [-version]
```

Example

```
rc -f <script_file_name> -logfile <results_file_name>
```

- Use *quit* or *exit* to close RC.
- Session data will be lost unless you have written it to disk.

Options

[-no_custom]: Only read master *.synth_init* file

[-E]: Exits on script error (When processing the -*files* option.)

[-unique]: Creates a unique log file and a unique command file for each session

[-files <string>]: Executes command file or files

[-execute <string>]: Command string to execute before processing the -*files* option

[-cmdfile <string>]: Specifies command logfile

[-logfile <string> | -nologfile]: Specifies a logfile, or not create one

[-gui]: Starts the graphical interface

[-lsf_cpus <integer>]: Number of LSF CPUs to use for super-threading

[-lsf_queue <string>]: Name of LSF queue

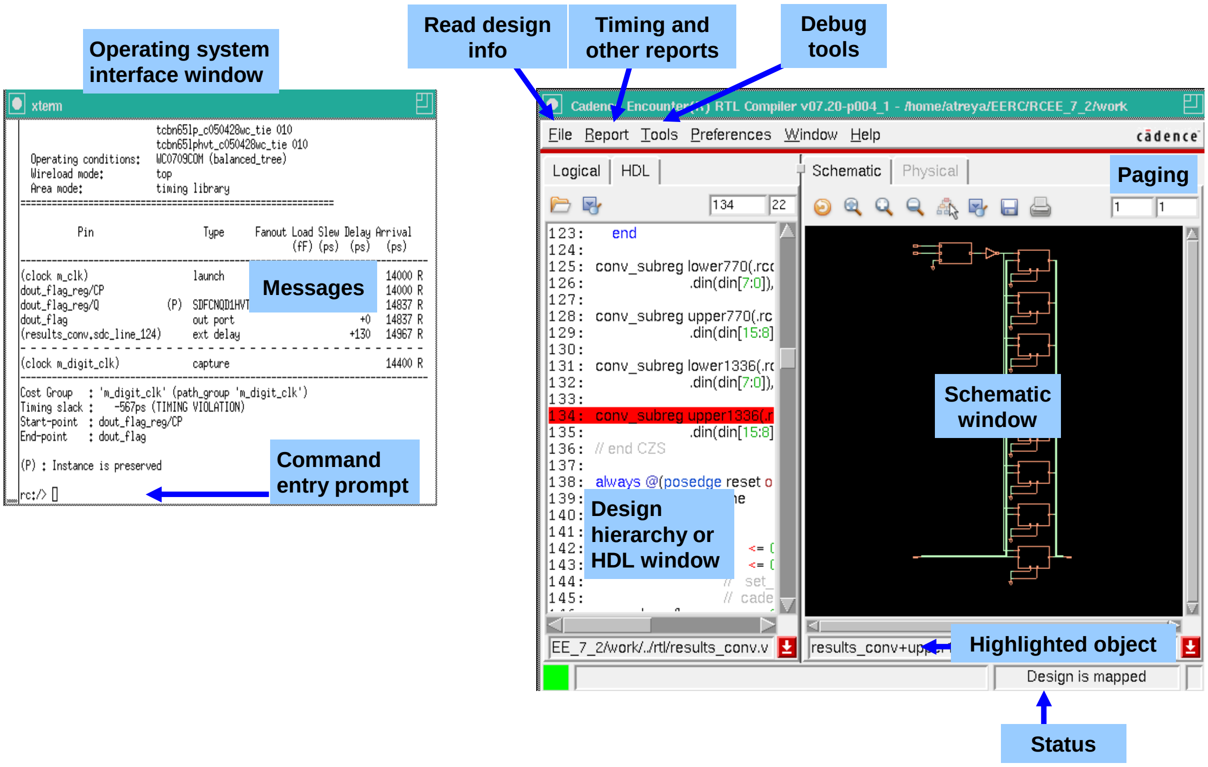
[-use_license <string>]: Specifies a license. When multiple licenses are specified, only the last one will be used.

[-queue]: Waits in a queue until a license is available

[-32 | -64]: Launches the 32 bit version or the 64 bit version

[-version]: Returns program version information

Graphical User Interface (GUI) Environment



```
                             tcbn65lp_c050428wc_tie 010
                             tcbn65lphvt_c050428wc_tie 010
  Operating conditions:      WC0709COM (balanced_tree)
  Wireload mode:             top
  Area mode:                 timing library
=============================================================

        Pin                      Type      Fanout Load Slew Delay Arrival
                                                  (fF) (ps)  (ps)   (ps)
------------------------------------------------------------------------------
(clock m_clk)                    launch                                14000 R
dout_flag_reg/CP                                                       14000 R
dout_flag_reg/Q              (P) SDFCNQD1HVT                           14837 R
dout_flag                        out port                         +0   14837 R
(results_conv.sdc_line_124)      ext delay                      +130   14967 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock m_digit_clk)              capture                               14400 R
------------------------------------------------------------------------------
Cost Group   : 'm_digit_clk' (path_group 'm_digit_clk')
Timing slack :    -567ps (TIMING VIOLATION)
Start-point  : dout_flag_reg/CP
End-point    : dout_flag

(P) : Instance is preserved

rc:/> 
```

```
    end

conv_subreg lower770(.rcc
                .din(din[7:0]),

conv_subreg upper770(.rc
                .din(din[15:8]

conv_subreg lower1336(.rc
                .din(din[7:0]),

conv_subreg upper1336(.r
                .din(din[15:8]
// end CZS

always @(posedge reset o
```

Operating system interface window
Messages
Command entry prompt
Read design info
Timing and other reports
Debug tools
xterm
Cadence Encounter(R) RTL Compiler v07.20-p004_1 - /home/atreya/EERC/RCEE_7_2/work
File Report Tools Preferences Window Help
cādence
Logical
HDL
134
22
Schematic
Physical
Paging
1
1
Schematic window
Design hierarchy or HDL window
EE_7_2/work/../rtl/results_conv.v
results_conv+upp
Highlighted object
Design is mapped
Status

Command Help

You can get help on all the commands by entering:

```
man command_name
```

or

```
help command_name
```

To view man pages from the UNIX shell, set your environment using:

```
setenv MANPATH $CDN_SYNTH_ROOT/share/synth/man
```

When you are unsure of a command, type the first few letters of the command and press the **Tab** key to display a list of commands that start with those letters. File completion also works from the command line.

Example

```
path_
```

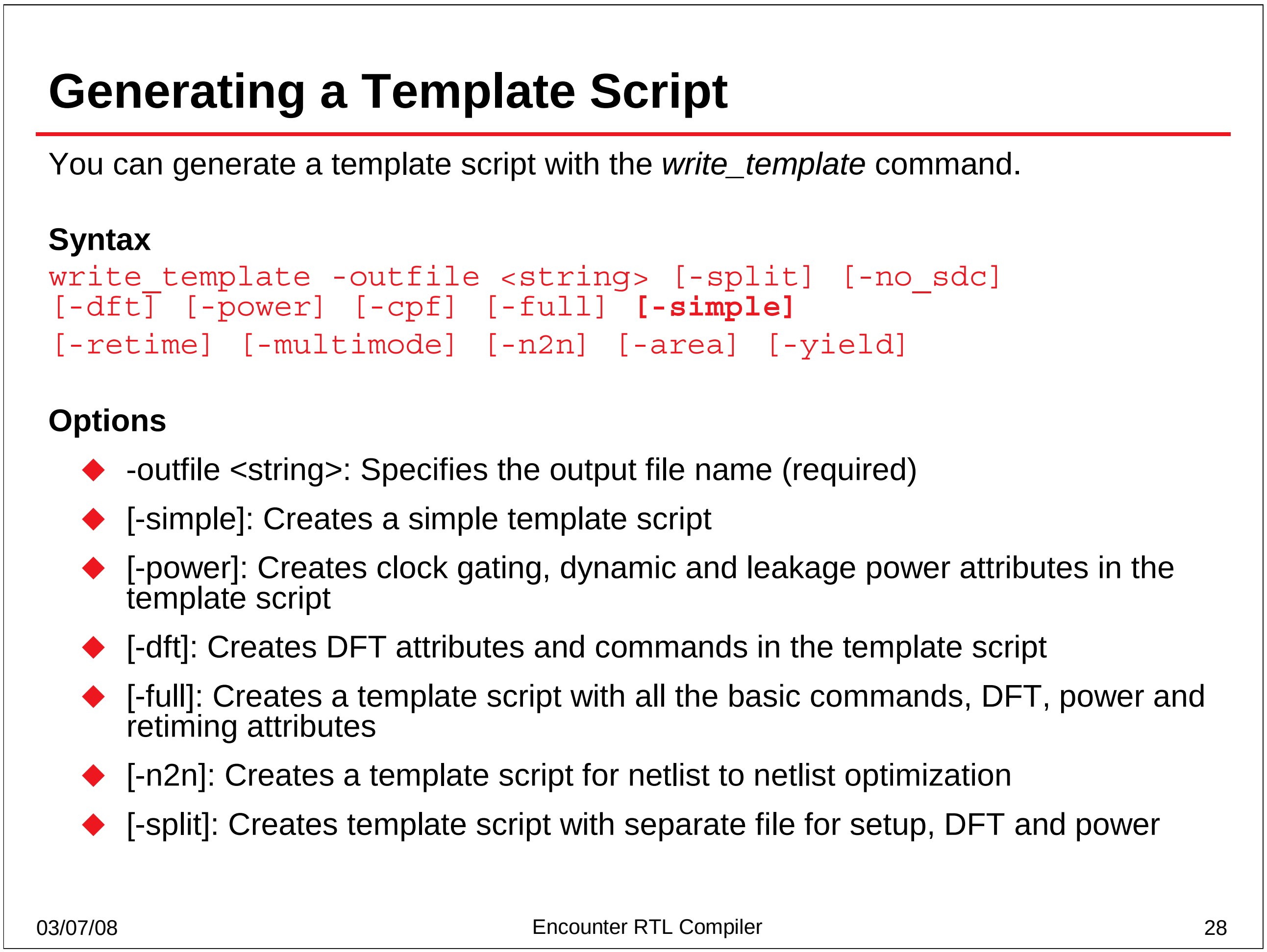
This command returns the following:

```
ambiguous "path_": path_adjust path_delay path_disable path_group
```

03/07/08 Encounter RTL Compiler 27

You can abbreviate the commands as long as there are no conflicting commands. If you use a conflicting command, the compiler gives an error message with a list of probable commands.

$*CDN_SYNTH_ROOT* points to the Cadence® installation of Encounter® RTL Compiler and must be included in your path.



Generating a Template Script

You can generate a template script with the *write_template* command.

Syntax

```
write_template -outfile <string> [-split] [-no_sdc]
[-dft] [-power] [-cpf] [-full] [-simple]
[-retime] [-multimode] [-n2n] [-area] [-yield]
```

Options

- -outfile <string>: Specifies the output file name (required)
- [-simple]: Creates a simple template script
- [-power]: Creates clock gating, dynamic and leakage power attributes in the template script
- [-dft]: Creates DFT attributes and commands in the template script
- [-full]: Creates a template script with all the basic commands, DFT, power and retiming attributes
- [-n2n]: Creates a template script for netlist to netlist optimization
- [-split]: Creates template script with separate file for setup, DFT and power

Other Options

- [-area]: Creates a template script for area optimization
- [-no_sdc]: Creates constraints in RC format
- [-msv]: Creates a template script for MSV flow in CPF format
- [-retime]: Creates retiming attributes and commands in the template script
- [-multimode]: Creates a template script for multimode analysis
- [-yield]: Creates a template script for yield optimization

Running Scripts

After RC starts, you can run a script by using one of the following methods.

- Use the source command:

  ```
  source <path>/<script_file_name>
  ```

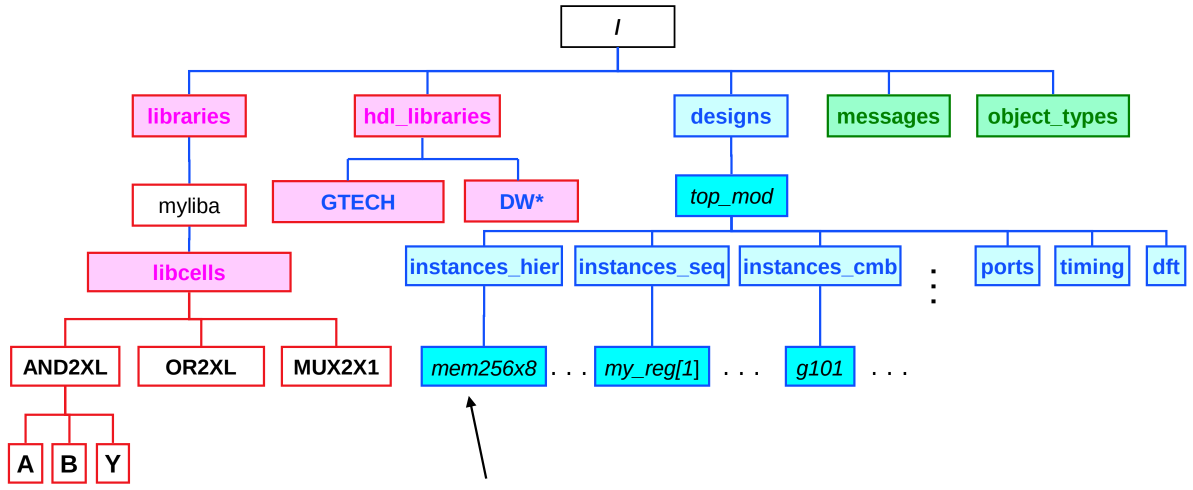
- Set the search paths so that the software can locate the files that RC uses.
  - To set the script search path, enter:

    ```
    set_attribute script_search_path <{ path1 path2 … }> /
    ```

  - Then, use:

    ```
    include script_file_name
    Or
    source script_file_name
    ```

Virtual Directory Structure

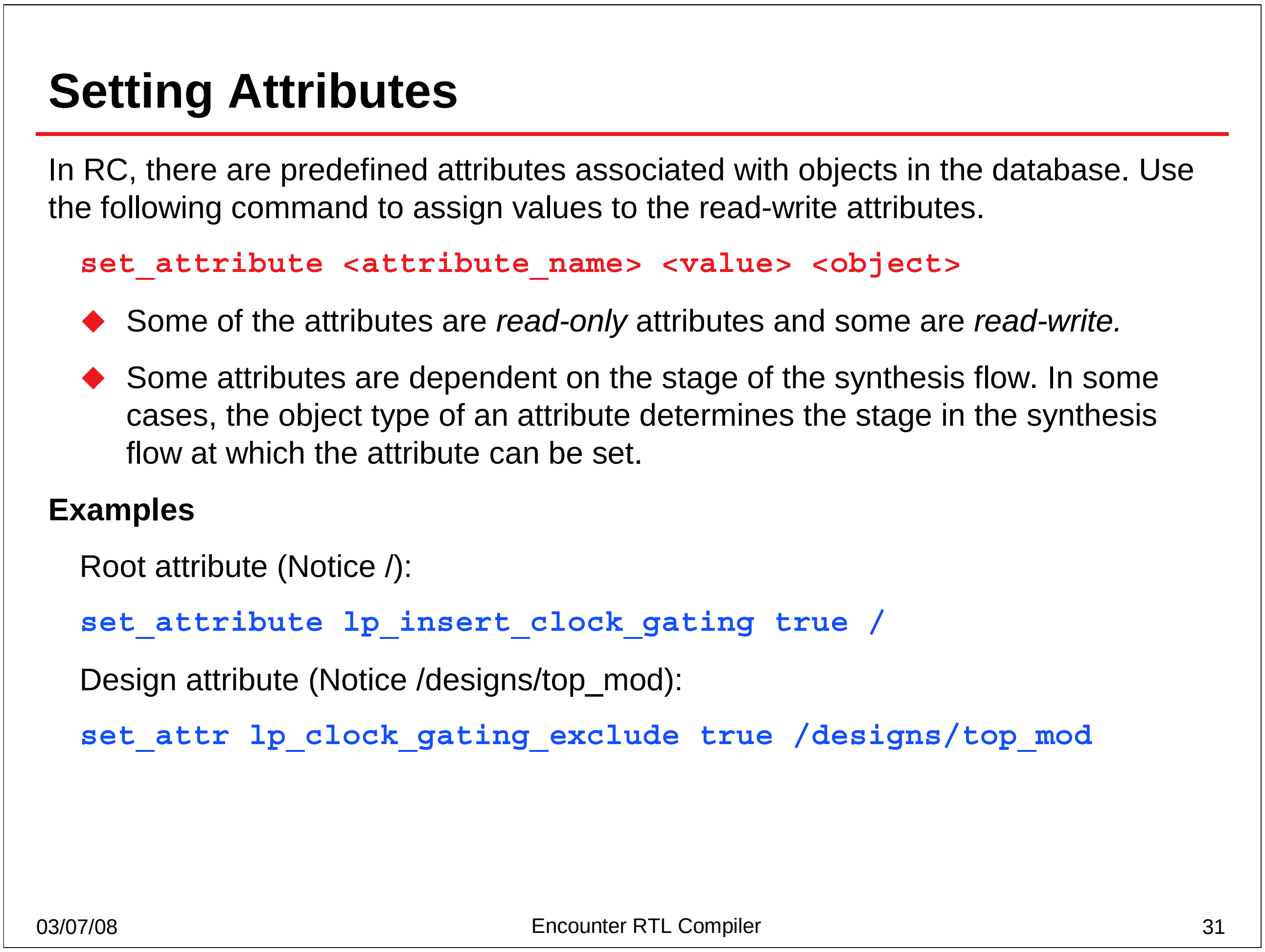


Directory structure maps to the design hierarchy.

The virtual directories contain objects and their attributes. The objects belong to object types such as designs, instances, clocks, and ports.

Many attributes affect the synthesis and optimization of these objects.

Tcl commands help navigate the database and modify objects. You can set the values on some of these attributes or create your own attributes.



Setting Attributes

In RC, there are predefined attributes associated with objects in the database. Use the following command to assign values to the read-write attributes.

```
set_attribute <attribute_name> <value> <object>
```

- Some of the attributes are *read-only* attributes and some are *read-write.*
- Some attributes are dependent on the stage of the synthesis flow. In some cases, the object type of an attribute determines the stage in the synthesis flow at which the attribute can be set.

Examples

Root attribute (Notice /):

```
set_attribute lp_insert_clock_gating true /
```

Design attribute (Notice /designs/top_mod):

```
set_attr lp_clock_gating_exclude true /designs/top_mod
```

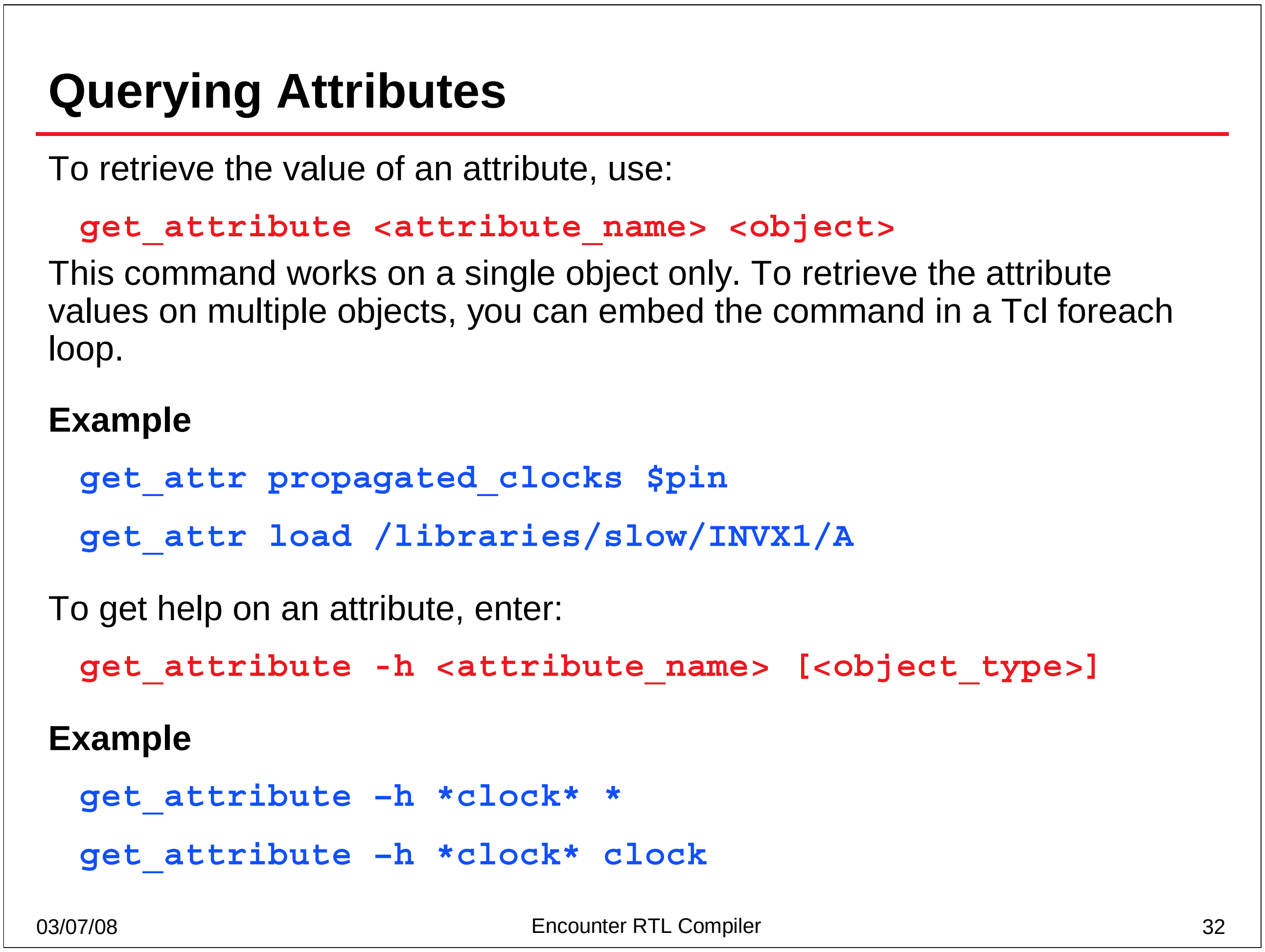
03/07/08 Encounter RTL Compiler 31

The *root_path* is the path of the object that the attribute is to be set to. Some attributes can only be set at the root (/) level. Some attributes can be set on the design objects only. Make sure all the attributes are properly set.

The compiler might not issue any warnings or error messages when the attribute is set on the wrong design object; for example, you defined the clock period on the *clk1* clock pin to 1000 ps, but you needed to set it on the *clk2* clock pin.

You can set your own attributes on the objects by providing a unique attribute name for the object type, by using the command:

```
set_user_attribute attribute_name attribute_value object
```



Querying Attributes

To retrieve the value of an attribute, use:

```
get_attribute <attribute_name> <object>
```

This command works on a single object only. To retrieve the attribute values on multiple objects, you can embed the command in a Tcl foreach loop.

Example

```
get_attr propagated_clocks $pin
get_attr load /libraries/slow/INVX1/A
```

To get help on an attribute, enter:

```
get_attribute -h <attribute_name> [<object_type>]
```

Example

```
get_attribute -h *clock* *
get_attribute -h *clock* clock
```

03/07/08 Encounter RTL Compiler 32

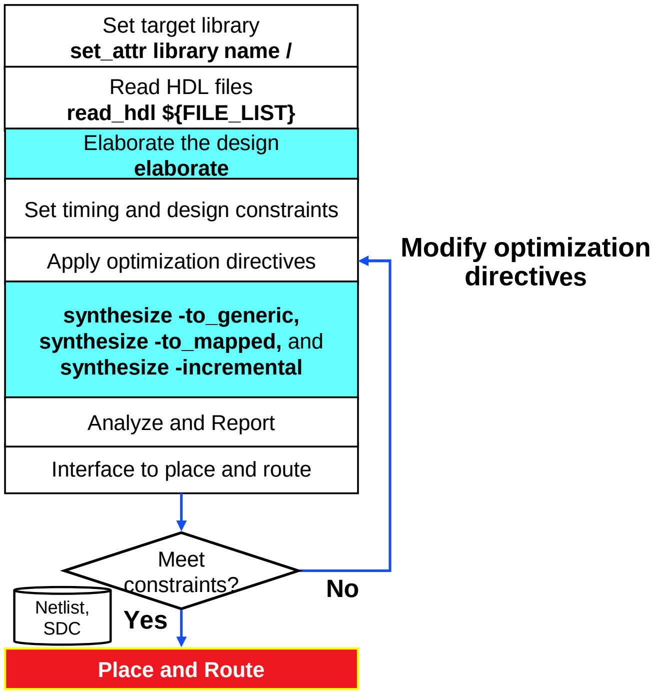
To reset the attribute to its original value, use the command:

```
reset_attribute [-quiet] attribute_name [object...]
```

To retrieve a user set attribute:

```
get_user_attribute attribute_name object
```

Basic Design Flow



Set target library
set_attr library name /

Read HDL files
read_hdl ${FILE_LIST}

Elaborate the design
elaborate

Set timing and design constraints

Apply optimization directives

synthesize -to_generic,
synthesize -to_mapped, and
synthesize -incremental

Analyze and Report

Interface to place and route

Meet constraints?

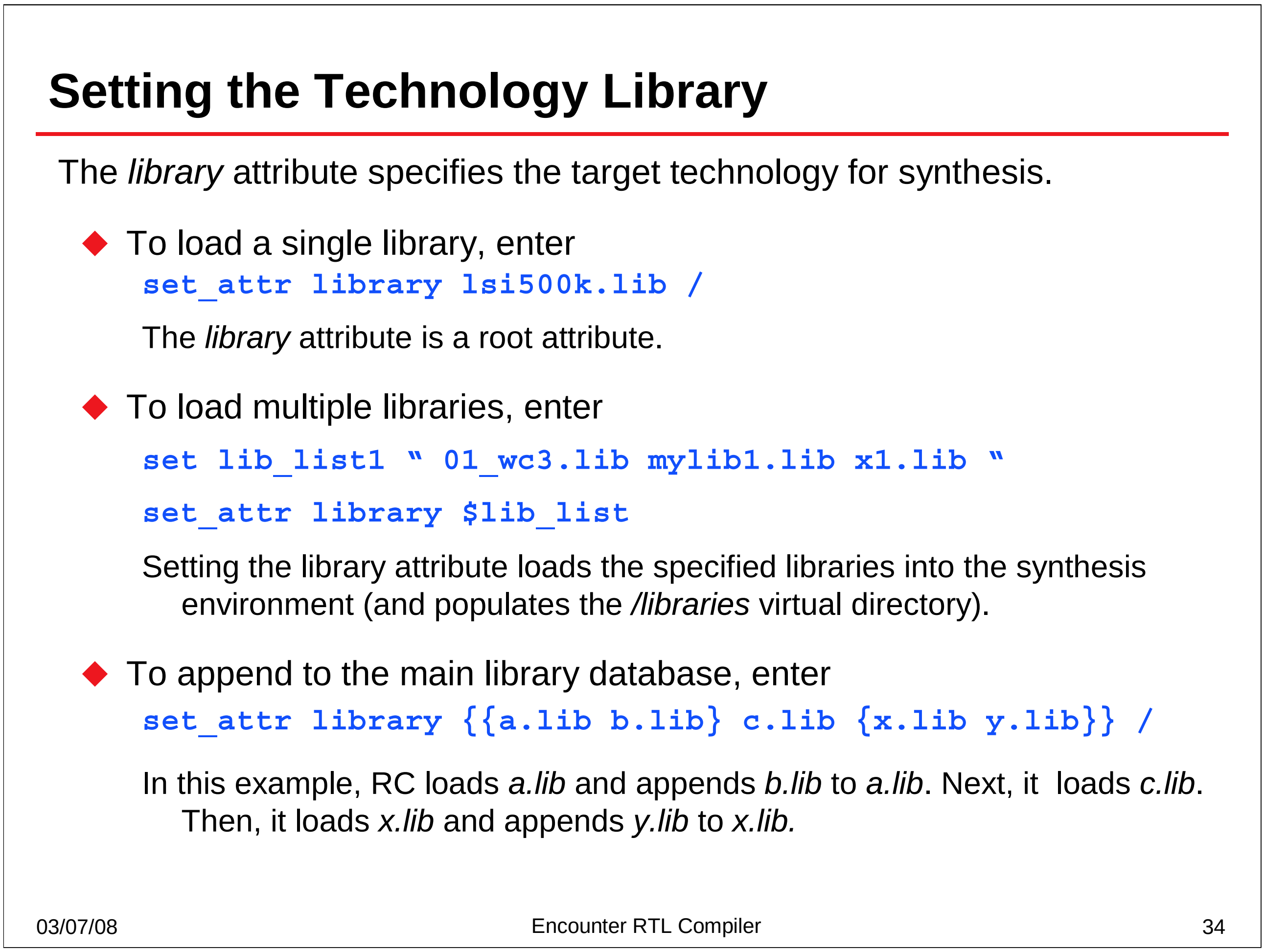
No

Modify optimization directives

Yes

Netlist, SDC

Place and Route



Setting the Technology Library

The *library* attribute specifies the target technology for synthesis.

- To load a single library, enter

  ```
  set_attr library lsi500k.lib /
  ```

  The *library* attribute is a root attribute.

- To load multiple libraries, enter

  ```
  set lib_list1 “ 01_wc3.lib mylib1.lib x1.lib “

  set_attr library $lib_list
  ```

  Setting the library attribute loads the specified libraries into the synthesis environment (and populates the */libraries* virtual directory).

- To append to the main library database, enter

  ```
  set_attr library {{a.lib b.lib} c.lib {x.lib y.lib}} /
  ```

  In this example, RC loads *a.lib* and appends *b.lib* to *a.lib*. Next, it loads *c.lib*. Then, it loads *x.lib* and appends *y.lib* to *x.lib.*

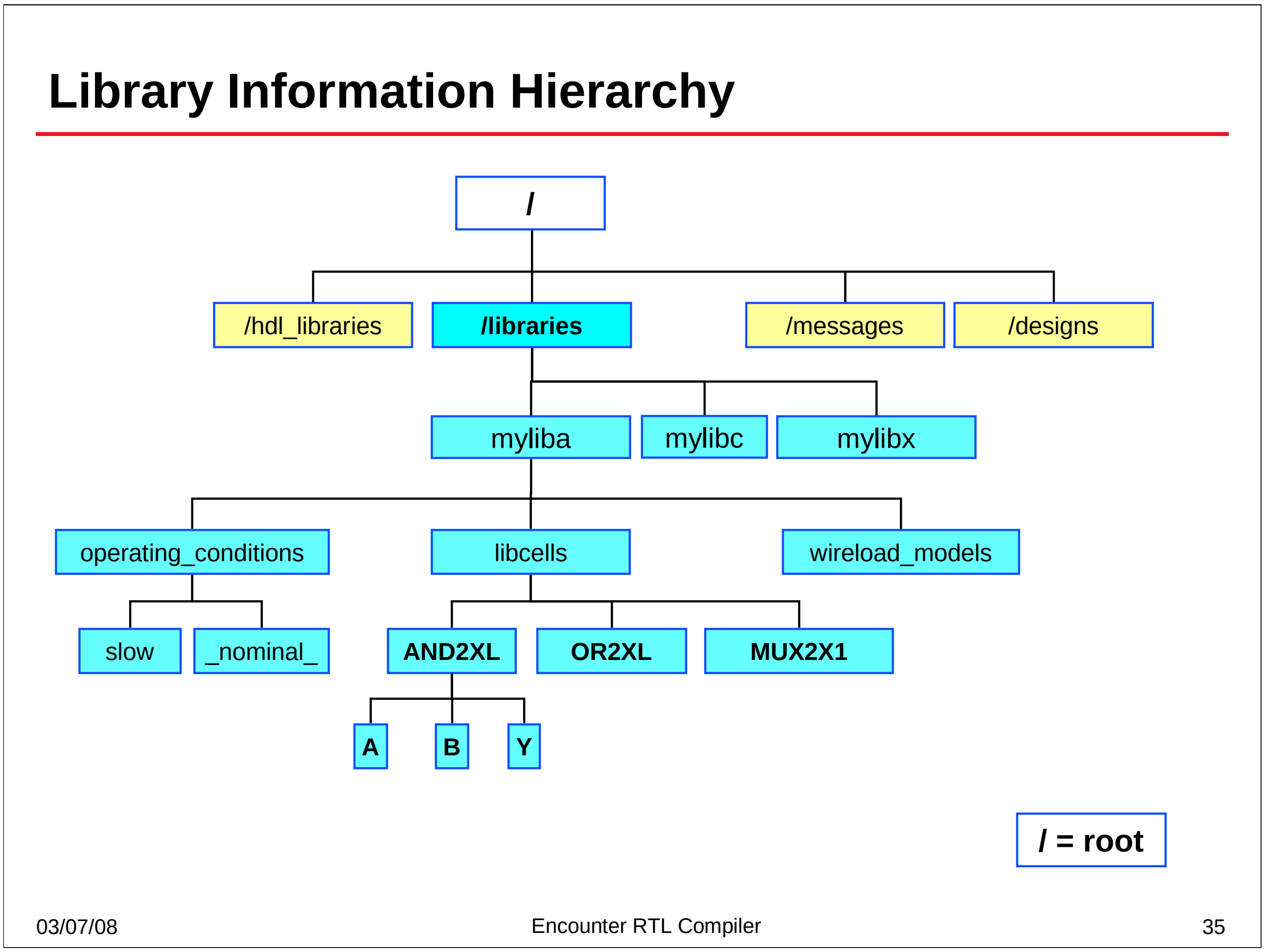
03/07/08 Encounter RTL Compiler 34

The source code for technology libraries is in the *.lib* format or in the *.lbr* binary format.

To set the library search path, enter:

```
set_attribute lib_search_path path /
```

A library includes control attributes, environment description, standard cell description, delay calculations, and delay models.



Library Information Hierarchy

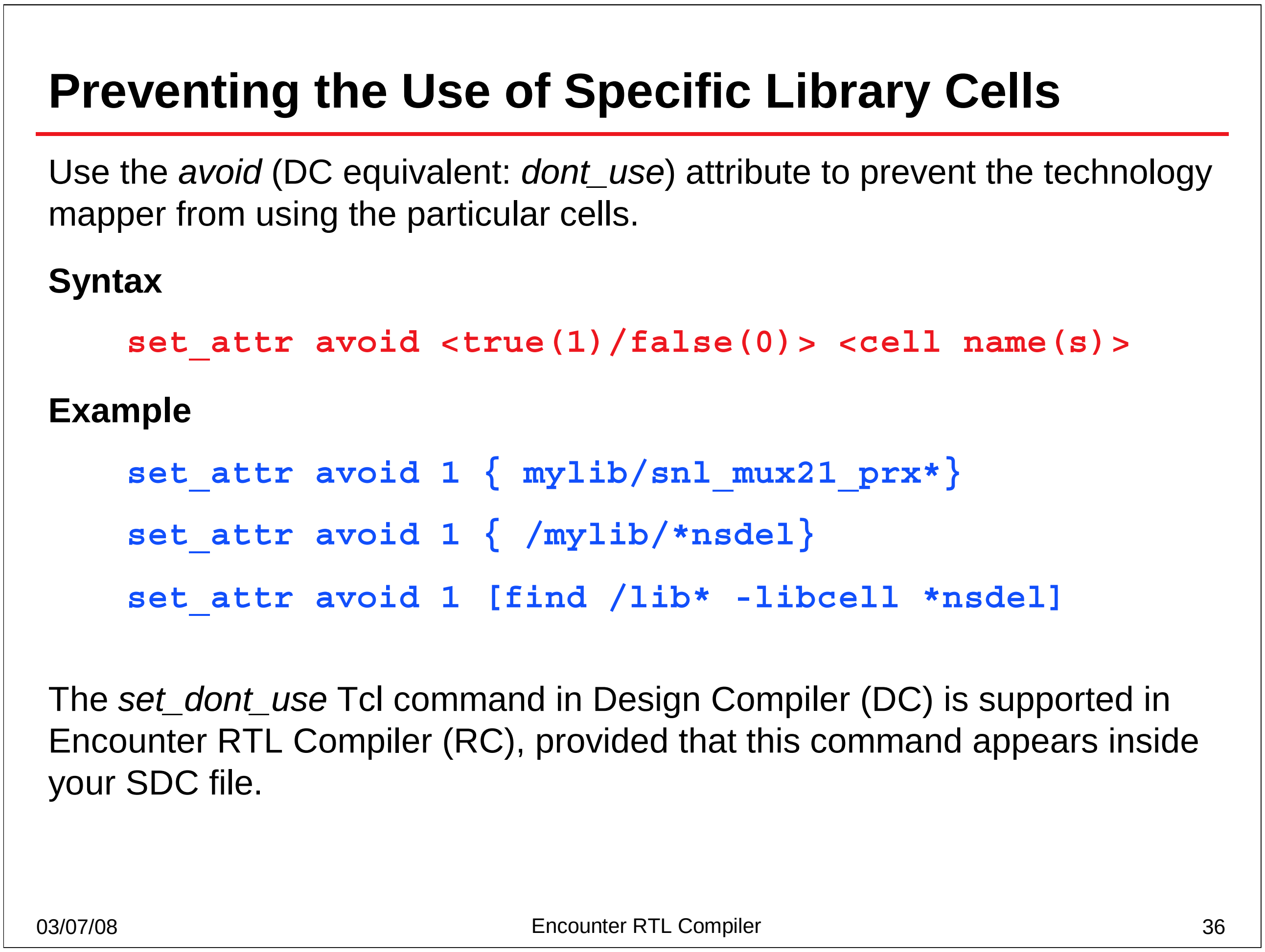
/
/hdl_libraries
/libraries
/messages
/designs
myliba
mylibc
mylibx
operating_conditions
libcells
wireload_models
slow
nominal
AND2XL
OR2XL
MUX2X1
A
B
Y

/ = root

03/07/08 Encounter RTL Compiler 35

When a library is read, the information is stored into the compiler memory as shown in this illustration.

There will be some changes in the library and design hierarchy when using multimode multisupply synthesis flow.



Preventing the Use of Specific Library Cells

Use the *avoid* (DC equivalent: *dont_use*) attribute to prevent the technology mapper from using the particular cells.

Syntax

```
set_attr avoid <true(1)/false(0)> <cell name(s)>
```

Example

```
set_attr avoid 1 { mylib/snl_mux21_prx*}
set_attr avoid 1 { /mylib/*nsdel}
set_attr avoid 1 [find /lib* -libcell *nsdel]
```

The *set_dont_use* Tcl command in Design Compiler (DC) is supported in Encounter RTL Compiler (RC), provided that this command appears inside your SDC file.

03/07/08 Encounter RTL Compiler 36

Preventing Optimization of Instances, Libcells

The *set_dont_touch* Tcl command in Design Compiler (DC) is supported in Encounter® RTL Compiler (RC), provided that this command appears inside your SDC file.

As with many DC Tcl commands, there is an RC equivalent attribute that you can include in your Tcl run script.

Syntax

```
set_attr preserve <true(1)/false(0)> <object name(s)>
```

Example

```
set_attr preserve 1 [find /lib*/lib1 -libcell *nsdel]
set dt_list [find /des* -subd acs4_*]
set_attr preserve true $dt_list
```

Wire Delay Estimation

Physical Layout Estimation (PLE)

- PLE uses actual design and physical library information.
- Dynamically calculates wire delays for different logic structures in the design.
- Correlates better with place and route.

set_attr lef_library <lef header>

set_attr cap_table_file <cap table>

set_attr interconnect_mode ple /

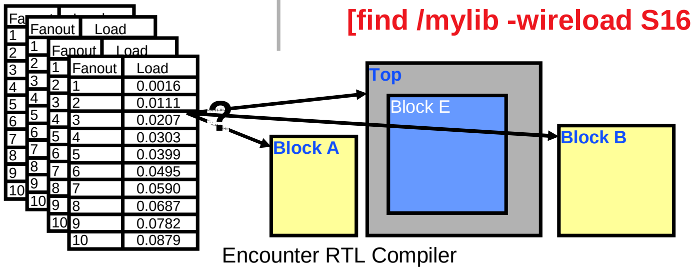
Wire-load Models

- Wire load models are statistical.
- Wire loads are calculated based on the nearest calibrated area.
- Selection of appropriate wire-load models for a design is tedious.
- Correlation is difficult even with custom wire-load models.

set_attr interconnect_mode wireload /

set_attr wireload_mode top /

set_attr force_wireload
[find /mylib -wireload S160K] /top

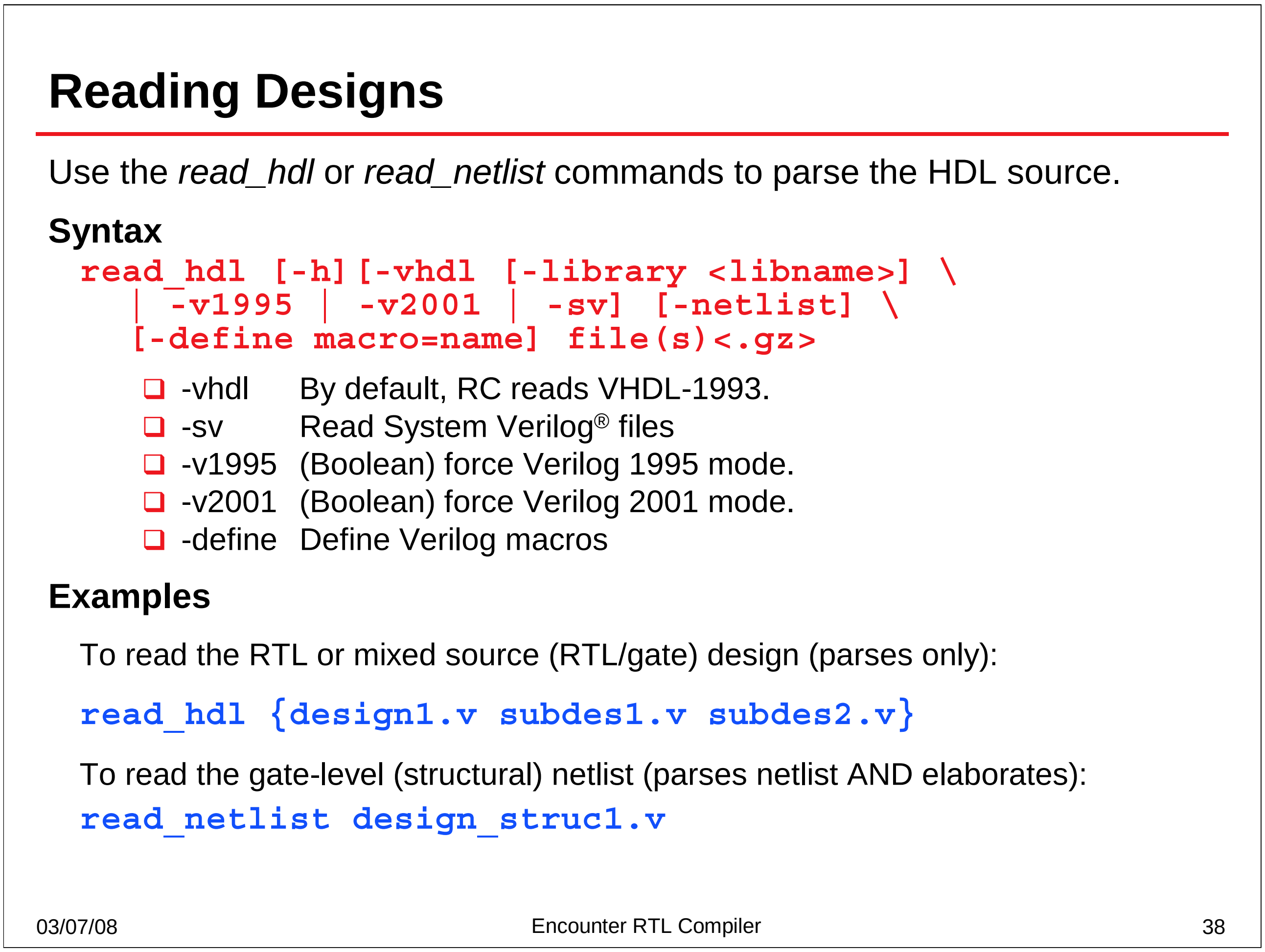


Fanout	Load
1	0.0016
2	0.0111
3	0.0207
4	0.0303
5	0.0399
6	0.0495
7	0.0590
8	0.0687
9	0.0782
10	0.0879

PLE is a physical modeling technique that bypasses wire loads for RTL synthesis optimization. In place of wire loads, the compiler generates an equation to model the wire delay.

- PLE removes the reliance on user-supplied WLMs.
- PLE is neither too optimistic or too pessimistic.

By default, the compiler selects the wire-load mode, and models automatically from the library.



Reading Designs

Use the *read_hdl* or *read_netlist* commands to parse the HDL source.

Syntax

```
read_hdl  [-h][-vhdl  [-library <libname>]  \
   | -v1995 | -v2001 | -sv]  [-netlist]  \
   [-define macro=name]  file(s)<.gz>
```

❑ -vhdl	By default, RC reads VHDL-1993.
❑ -sv	Read System Verilog® files
❑ -v1995	(Boolean) force Verilog 1995 mode.
❑ -v2001	(Boolean) force Verilog 2001 mode.
❑ -define	Define Verilog macros

Examples

To read the RTL or mixed source (RTL/gate) design (parses only):

```
read_hdl {design1.v subdes1.v subdes2.v}
```

To read the gate-level (structural) netlist (parses netlist AND elaborates):

```
read_netlist design_struc1.v
```

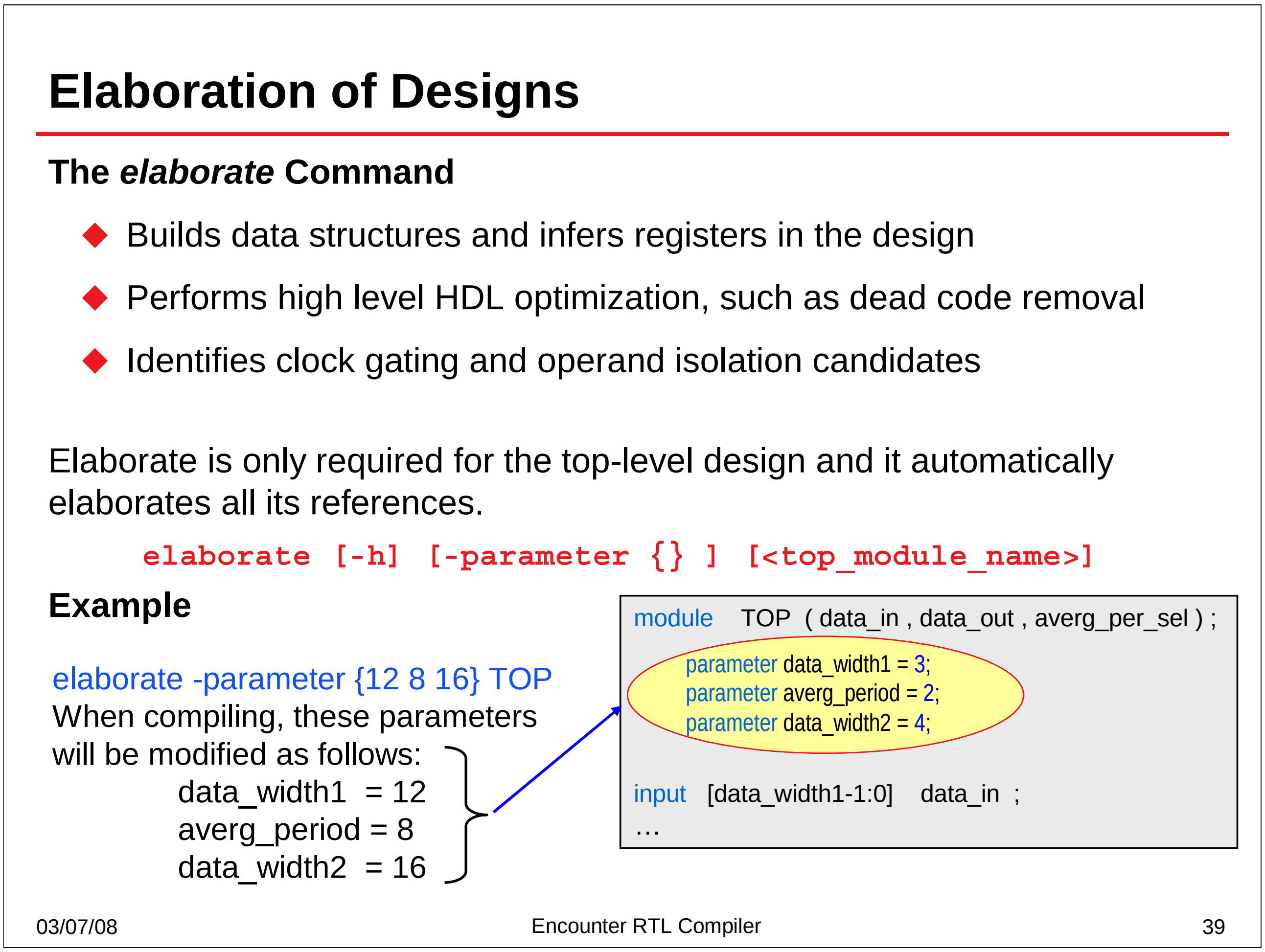
To set the HDL search path, enter the following:

```
set_attribute hdl_search_path path /
```

Alternatively, to read a VHDL design, enter the following:

```
set_attr hdl_language vhdl /
set_attr hdl_vhdl_read_version [1987|1993] /
read_hdl -vhdl design.v -library library_name
```

You must load all submodules and the top-level module into the compiler.



Elaboration of Designs

The *elaborate* Command

- Builds data structures and infers registers in the design
- Performs high level HDL optimization, such as dead code removal
- Identifies clock gating and operand isolation candidates

Elaborate is only required for the top-level design and it automatically elaborates all its references.

```
elaborate [-h] [-parameter {} ] [<top_module_name>]
```

Example

elaborate -parameter {12 8 16} TOP
When compiling, these parameters will be modified as follows:

data_width1 = 12
averg_period = 8
data_width2 = 16

```
module   TOP ( data_in , data_out , averg_per_sel ) ;
        parameter data_width1 = 3;
        parameter averg_period = 2;
        parameter data_width2 = 4;

input  [data_width1-1:0]   data_in  ;
…
```

-h: Help with the elaborate command

-parameters: Integer list of design parameters

[-libpath path]: Specifies the search path for unresolved Verilog module instances

[-libext extension]: Specifies the extension of the Verilog library

(string): Top-level module name

Before elaborating a design

- Read in all the designs.
- Set the general compiler environment.

After elaboration finishes, the software reports

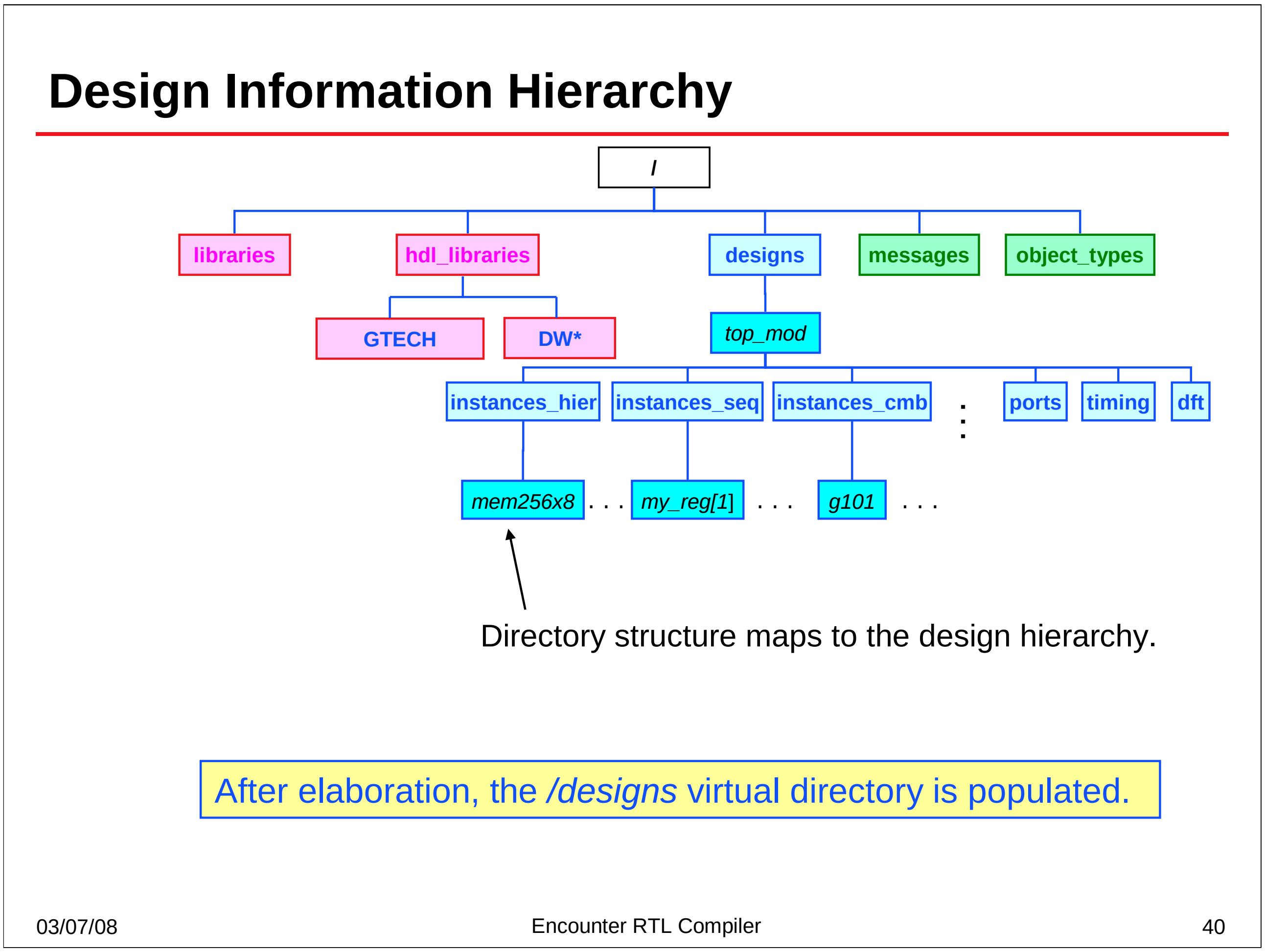
- Unresolved references, that is, instances found with no corresponding module or library cell
- Semantic problems, including:

  Read before write

  Unused ports, inconsistent resets

Use the *max_print* attribute to limit messages in the log file.

```
set_attr max_print 1 [find / -message ELAB-VLOG-16]
```



Design Information Hierarchy

/

libraries

hdl_libraries

GTECH

DW*

designs

top_mod

instances_hier

instances_seq

instances_cmb

ports

timing

dft

mem256x8 . . . my_reg[1] . . . g101 . . .

messages

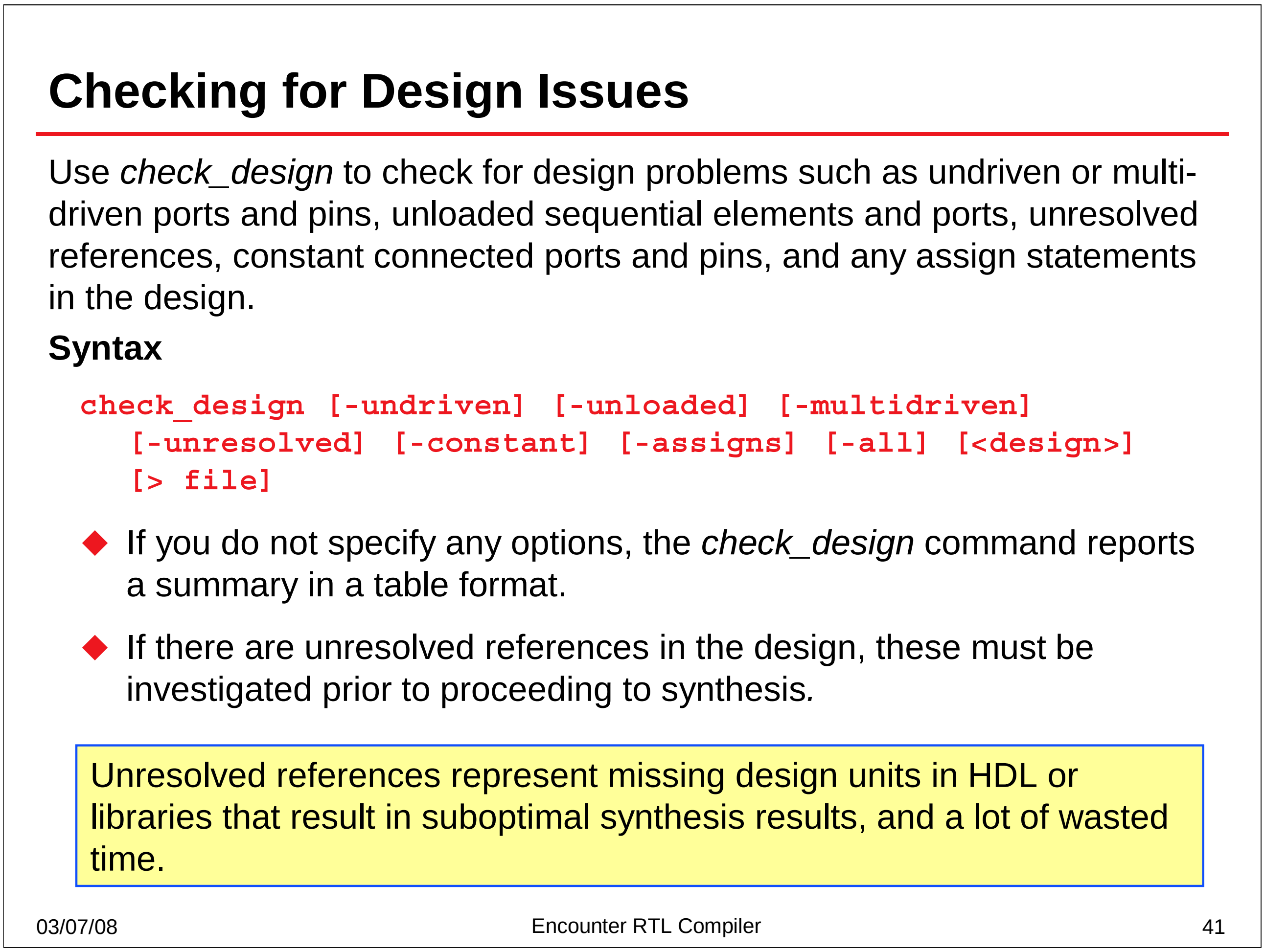
object_types

Directory structure maps to the design hierarchy.

After elaboration, the */designs* virtual directory is populated.

03/07/08 Encounter RTL Compiler 40

After a design is read and elaborated, the information is stored in the compiler memory as shown in this illustration.



Checking for Design Issues

Use *check_design* to check for design problems such as undriven or multi-driven ports and pins, unloaded sequential elements and ports, unresolved references, constant connected ports and pins, and any assign statements in the design.

Syntax

```
check_design [-undriven] [-unloaded] [-multidriven]
   [-unresolved] [-constant] [-assigns] [-all] [<design>]
   [> file]
```

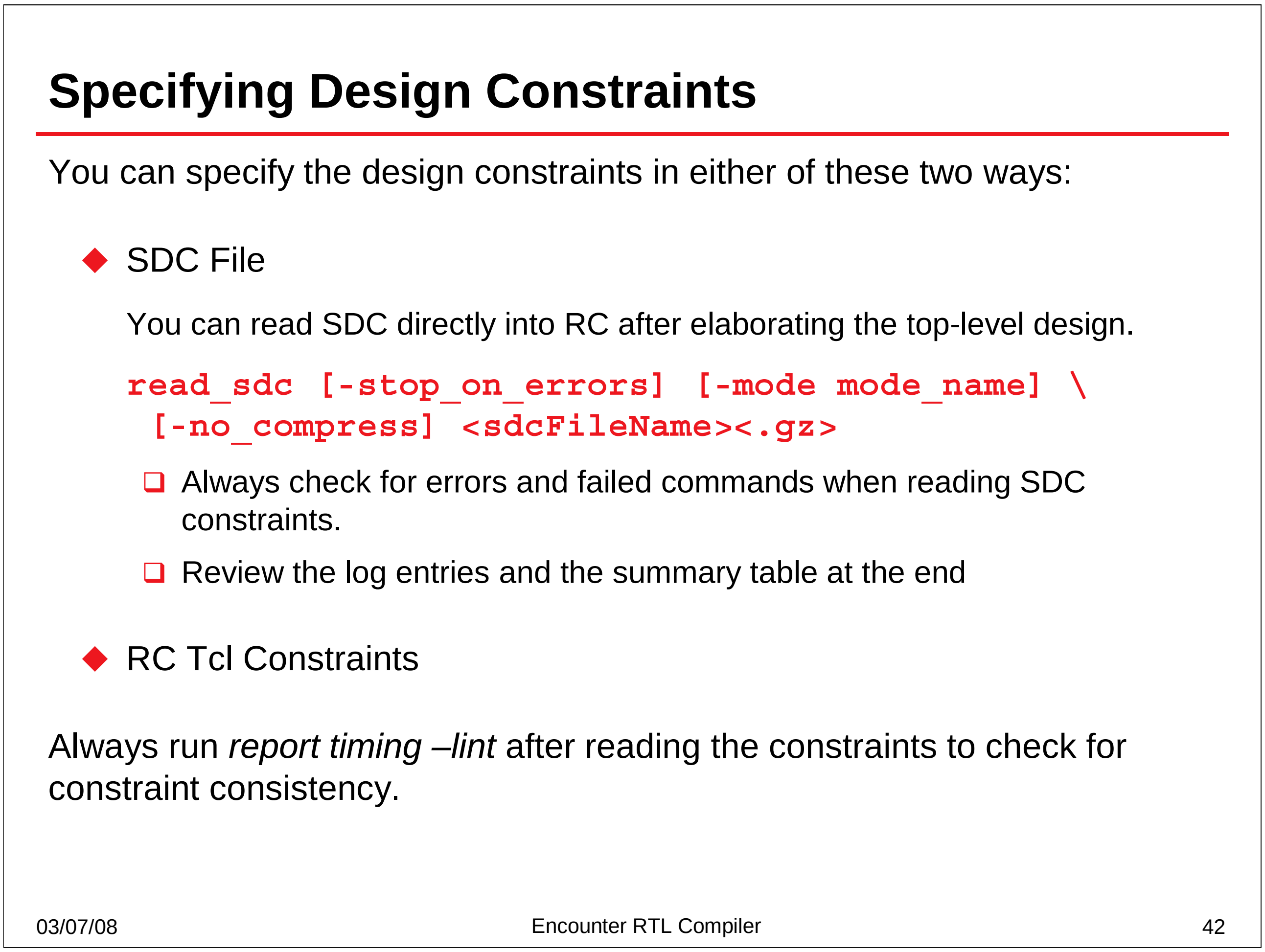
- If you do not specify any options, the *check_design* command reports a summary in a table format.
- If there are unresolved references in the design, these must be investigated prior to proceeding to synthesis.

Unresolved references represent missing design units in HDL or libraries that result in suboptimal synthesis results, and a lot of wasted time.

03/07/08 Encounter RTL Compiler 41

Check Design Report Summary

Name	Total
Unresolved References	0
Empty Modules	0
Unloaded Port(s)	0
Unloaded Sequential Pin(s)	15
Assigns	548
Undriven Port(s)	0
Undriven Leaf Pin(s)	0
Undriven hierarchical pin(s)	15
Multidriven Port(s)	0
Multidriven Leaf Pin(s)	0
Multidriven hierarchical Pin(s)	0
Constant Port(s)	0
Constant Leaf Pin(s)	34
Constant hierarchical Pin(s)	1535



Specifying Design Constraints

You can specify the design constraints in either of these two ways:

- SDC File

  You can read SDC directly into RC after elaborating the top-level design.

  ```
  read_sdc [-stop_on_errors] [-mode mode_name] \
   [-no_compress] <sdcFileName><.gz>
  ```

  - Always check for errors and failed commands when reading SDC constraints.
  - Review the log entries and the summary table at the end

- RC Tcl Constraints

Always run *report timing –lint* after reading the constraints to check for constraint consistency.

03/07/08 Encounter RTL Compiler 42

Use an SDC file when you import a design from other synthesis environments like BuildGates® Extreme or Design Compiler.

When using an SDC file, the capacitance specified in picofarads *(pF)* (SDC unit) is converted to femtofarads *(fF)* (RC unit) and time specified in *ns* is converted to *ps*.

You can use SDC commands interactively by using the **dc::** as a prefix. But when mixing DC commands and Encounter RTL Compiler (RC) commands, be very careful with the units of capacitance and delay.

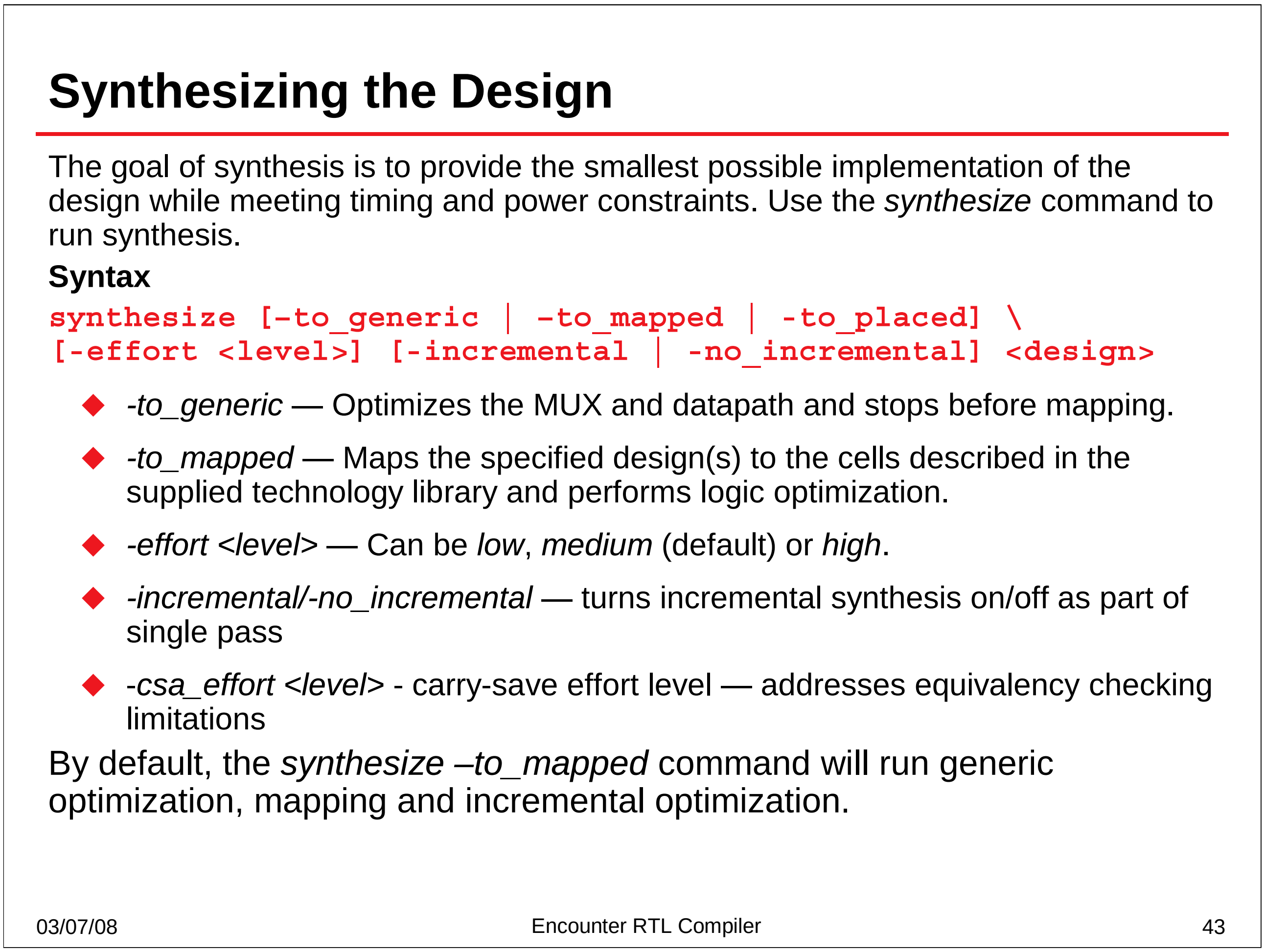
```
dc::set_load [get_attr load slow/INVX1/A] [dc::all_outputs]
```

In this case, the capacitance on all outputs will be off by a factor of 1000 because of the conversion from pF to fF. For example, *get_attr* , which is an RC command returns a load value of *10000 fF* from INVX1/A. The DC command *set_load*, is expecting loads in DC units (*pF*), and sets a load of *10000 pF* on all the outputs.

Instead, use separate commands with conversion factor included.

```
set loadIV [ expr [get_attr load slow/INVX1/A]/1000]
dc:: set_load loadIV [dc::all_outputs]
```

Remember to use **dc::** with every command, even if used recursively (command within a command).



Synthesizing the Design

The goal of synthesis is to provide the smallest possible implementation of the design while meeting timing and power constraints. Use the *synthesize* command to run synthesis.

Syntax

```
synthesize [-to_generic | -to_mapped | -to_placed] \
[-effort <level>] [-incremental | -no_incremental] <design>
```

- *-to_generic* — Optimizes the MUX and datapath and stops before mapping.
- *-to_mapped* — Maps the specified design(s) to the cells described in the supplied technology library and performs logic optimization.
- *-effort <level>* — Can be *low*, *medium* (default) or *high*.
- *-incremental/-no_incremental* — turns incremental synthesis on/off as part of single pass
- *-csa_effort <level>* - carry-save effort level — addresses equivalency checking limitations

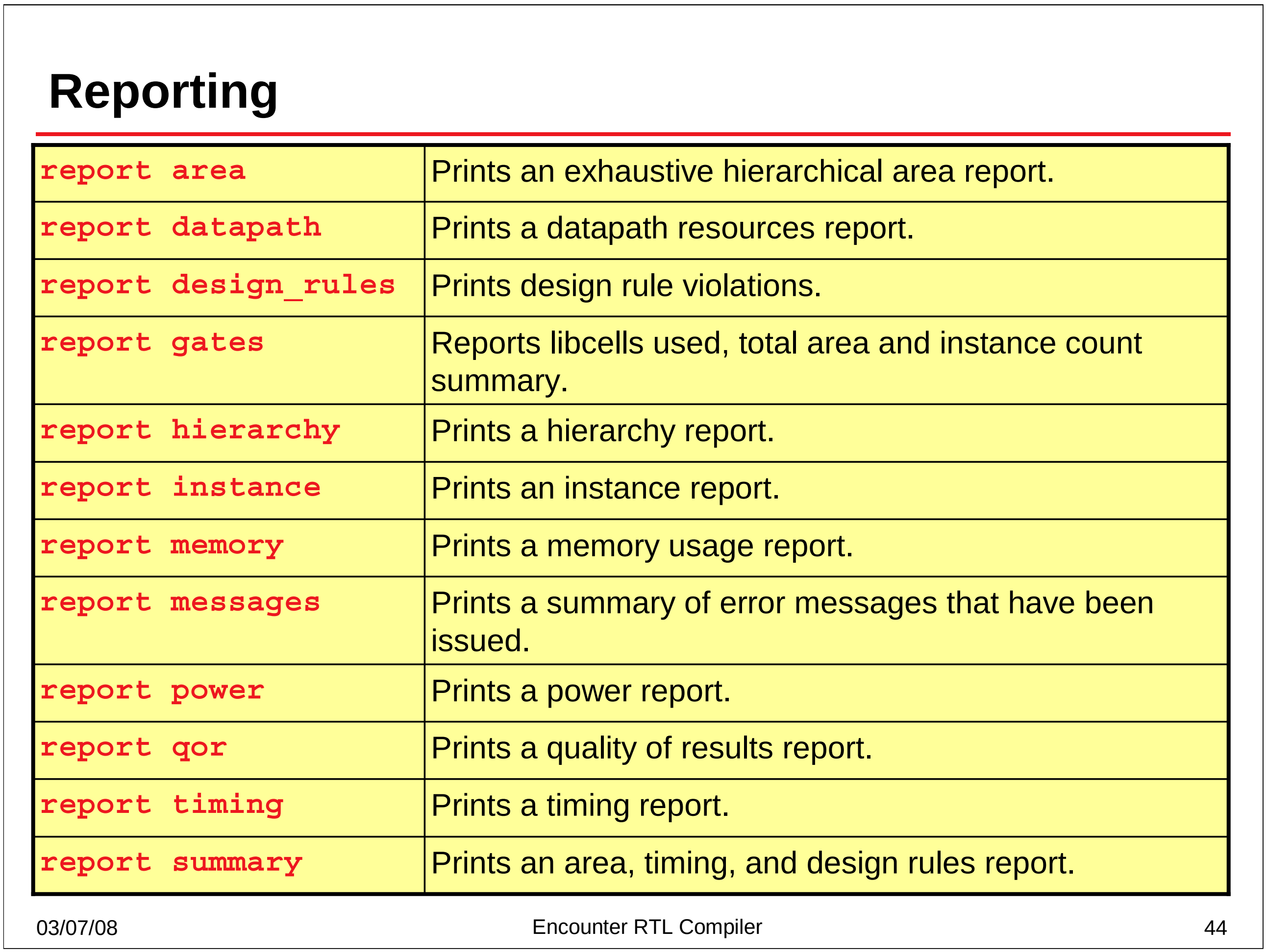
By default, the *synthesize –to_mapped* command will run generic optimization, mapping and incremental optimization.

Example

To perform timing-driven RTL optimizations (without targeting any specific technology cells), run this command:

```
synthesize -to_generic -effort high
```

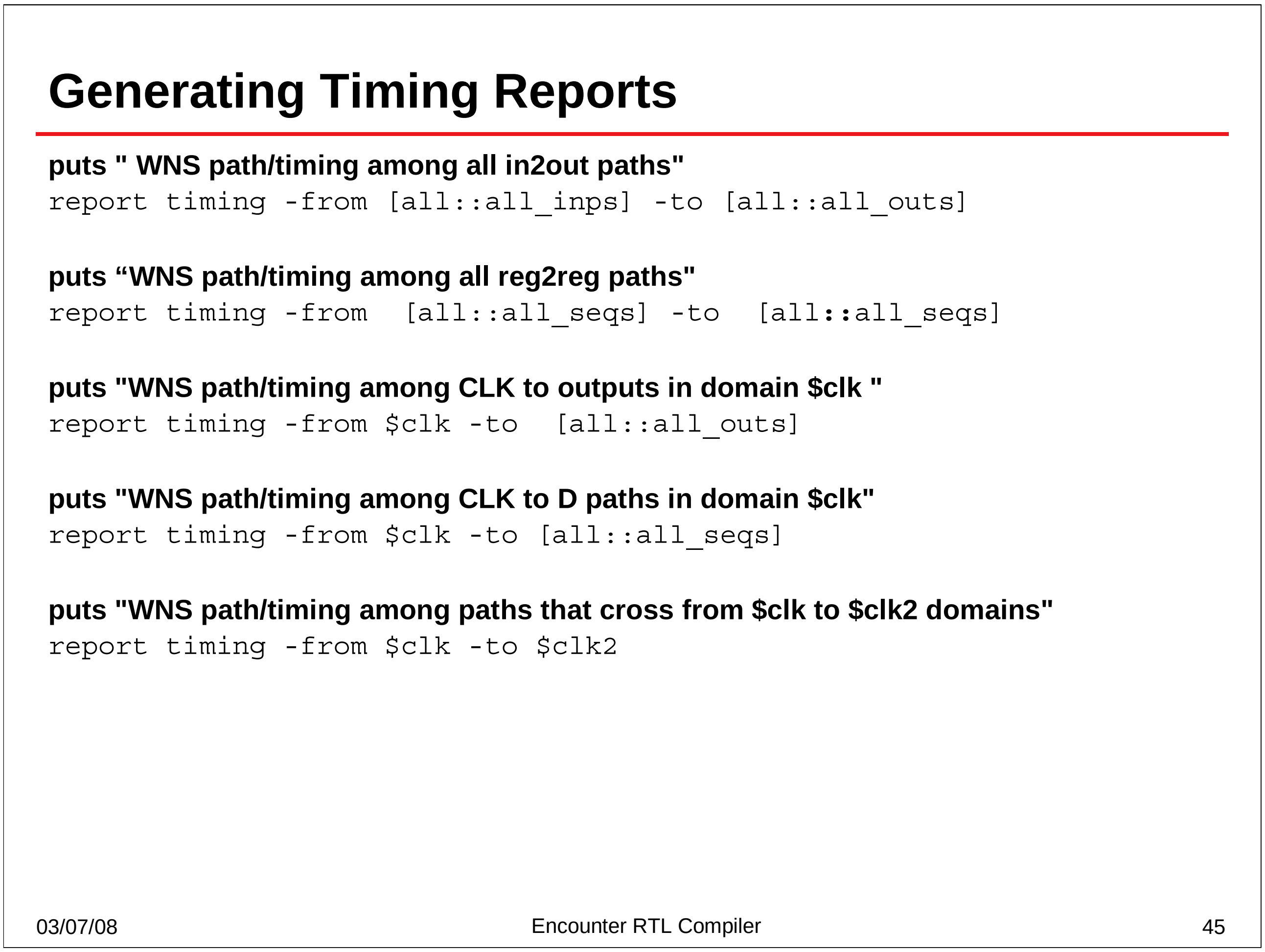
The generic netlist contains technology-independent components that are generated within the Encounter RTL Complier.



Reporting

`report area`	Prints an exhaustive hierarchical area report.
`report datapath`	Prints a datapath resources report.
`report design_rules`	Prints design rule violations.
`report gates`	Reports libcells used, total area and instance count summary.
`report hierarchy`	Prints a hierarchy report.
`report instance`	Prints an instance report.
`report memory`	Prints a memory usage report.
`report messages`	Prints a summary of error messages that have been issued.
`report power`	Prints a power report.
`report qor`	Prints a quality of results report.
`report timing`	Prints a timing report.
`report summary`	Prints an area, timing, and design rules report.

03/07/08 Encounter RTL Compiler 44



Generating Timing Reports

puts " WNS path/timing among all in2out paths"

```
report timing -from [all::all_inps] -to [all::all_outs]
```

puts “WNS path/timing among all reg2reg paths"

```
report timing -from  [all::all_seqs] -to  [all::all_seqs]
```

puts "WNS path/timing among CLK to outputs in domain $clk "

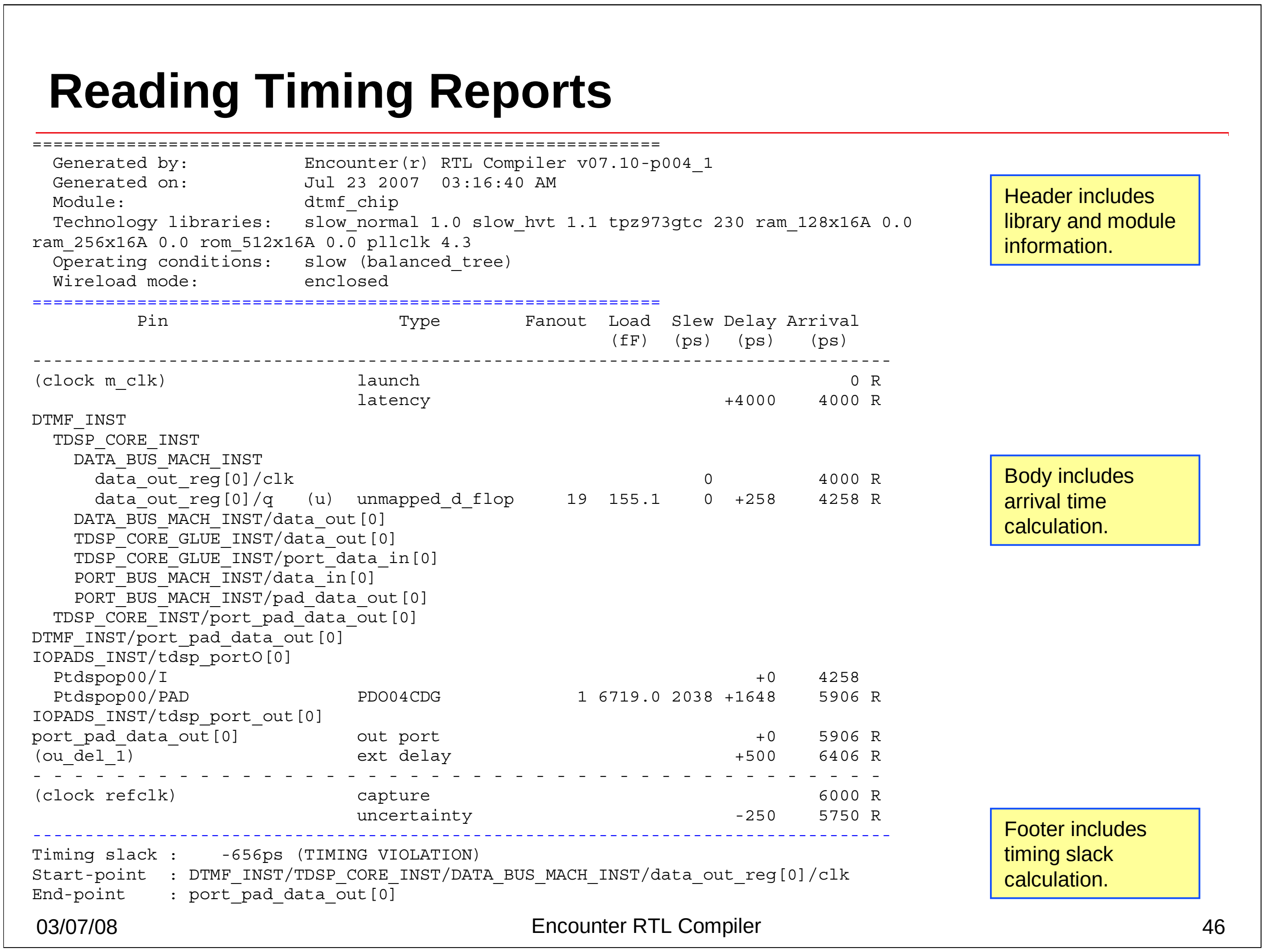
```
report timing -from $clk -to  [all::all_outs]
```

puts "WNS path/timing among CLK to D paths in domain $clk"

```
report timing -from $clk -to [all::all_seqs]
```

puts "WNS path/timing among paths that cross from $clk to $clk2 domains"

```
report timing -from $clk -to $clk2
```



Reading Timing Reports

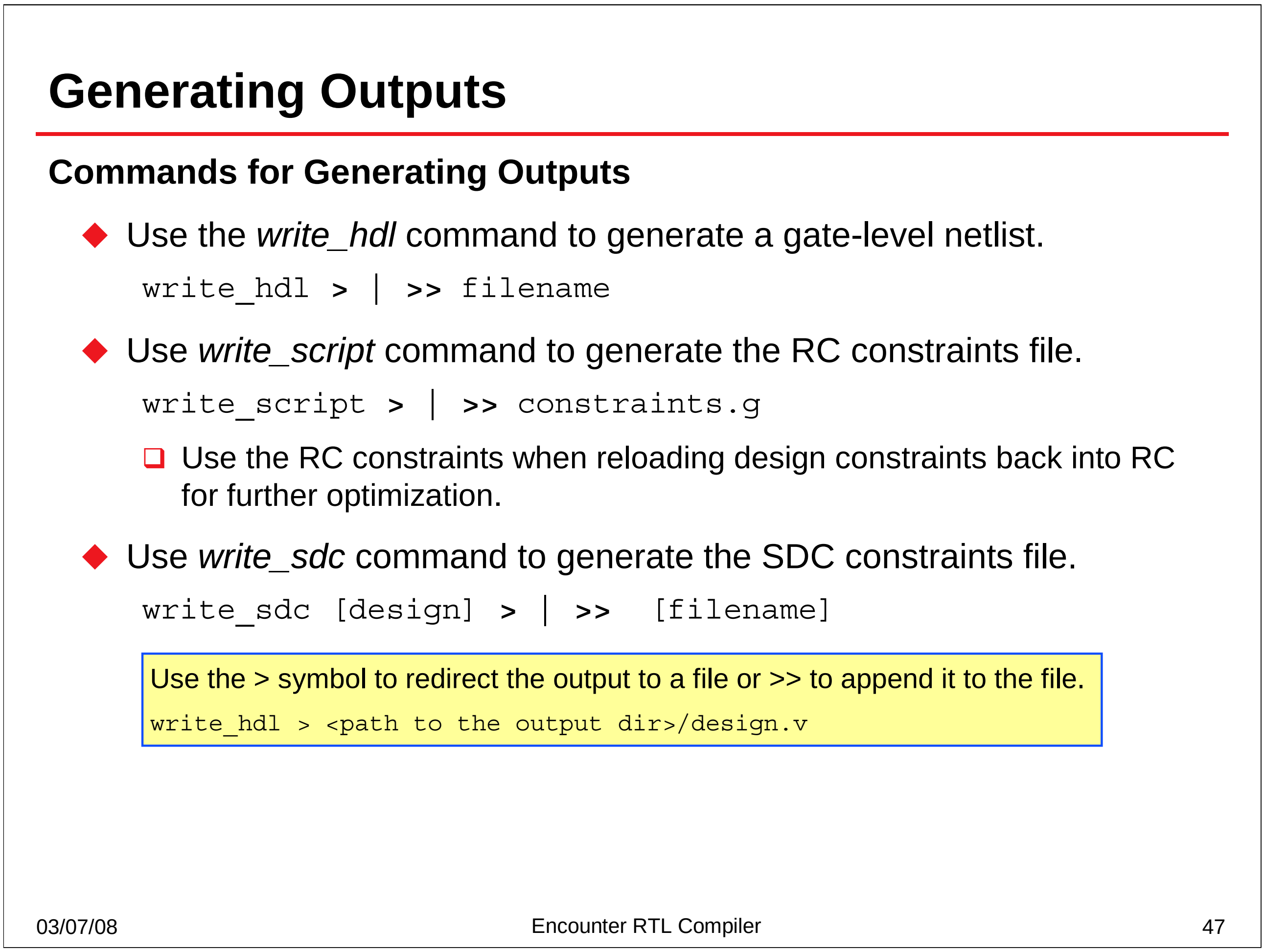
```
============================================================
  Generated by:           Encounter(r) RTL Compiler v07.10-p004_1
  Generated on:           Jul 23 2007  03:16:40 AM
  Module:                 dtmf_chip
  Technology libraries:   slow_normal 1.0 slow_hvt 1.1 tpz973gtc 230 ram_128x16A 0.0
ram_256x16A 0.0 rom_512x16A 0.0 pllclk 4.3
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
============================================================
          Pin                      Type        Fanout  Load  Slew Delay Arrival
                                                       (fF)  (ps)  (ps)    (ps)
----------------------------------------------------------------------------------
(clock m_clk)                  launch                                         0 R
                               latency                            +4000    4000 R
DTMF_INST
  TDSP_CORE_INST
    DATA_BUS_MACH_INST
      data_out_reg[0]/clk                                       0          4000 R
      data_out_reg[0]/q   (u)  unmapped_d_flop     19  155.1    0  +258    4258 R
    DATA_BUS_MACH_INST/data_out[0]
    TDSP_CORE_GLUE_INST/data_out[0]
    TDSP_CORE_GLUE_INST/port_data_in[0]
    PORT_BUS_MACH_INST/data_in[0]
    PORT_BUS_MACH_INST/pad_data_out[0]
  TDSP_CORE_INST/port_pad_data_out[0]
DTMF_INST/port_pad_data_out[0]
IOPADS_INST/tdsp_portO[0]
  Ptdspop00/I                                                         +0    4258
  Ptdspop00/PAD                PDO04CDG             1 6719.0 2038 +1648    5906 R
IOPADS_INST/tdsp_port_out[0]
port_pad_data_out[0]           out port                               +0    5906 R
(ou_del_1)                     ext delay                            +500    6406 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock refclk)                 capture                                     6000 R
                               uncertainty                          -250    5750 R
----------------------------------------------------------------------------------
Timing slack :    -656ps (TIMING VIOLATION)
Start-point  : DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/data_out_reg[0]/clk
End-point    : port_pad_data_out[0]
```

Header includes library and module information.

Body includes arrival time calculation.

Footer includes timing slack calculation.

03/07/08 Encounter RTL Compiler 46



Generating Outputs

Commands for Generating Outputs

- Use the *write_hdl* command to generate a gate-level netlist.

  ```
  write_hdl > | >> filename
  ```

- Use *write_script* command to generate the RC constraints file.

  ```
  write_script > | >> constraints.g
  ```

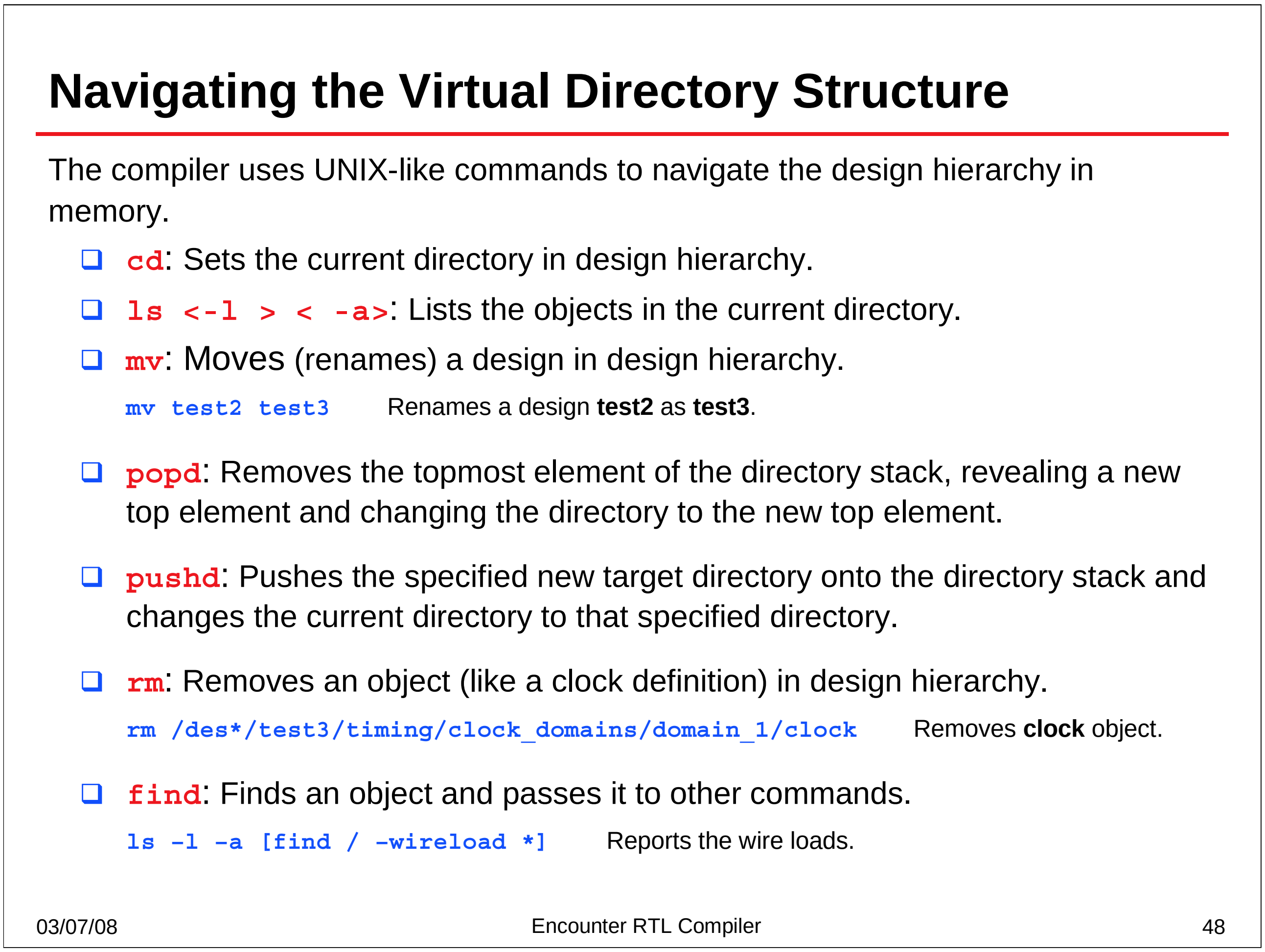
  - Use the RC constraints when reloading design constraints back into RC for further optimization.

- Use *write_sdc* command to generate the SDC constraints file.

  ```
  write_sdc [design] > | >>  [filename]
  ```

Use the > symbol to redirect the output to a file or >> to append it to the file.

```
write_hdl > <path to the output dir>/design.v
```



Navigating the Virtual Directory Structure

The compiler uses UNIX-like commands to navigate the design hierarchy in memory.

- `cd`: Sets the current directory in design hierarchy.
- `ls <-l > < -a>`: Lists the objects in the current directory.
- `mv`: Moves (renames) a design in design hierarchy.

  `mv test2 test3`   Renames a design **test2** as **test3**.

- `popd`: Removes the topmost element of the directory stack, revealing a new top element and changing the directory to the new top element.
- `pushd`: Pushes the specified new target directory onto the directory stack and changes the current directory to that specified directory.
- `rm`: Removes an object (like a clock definition) in design hierarchy.

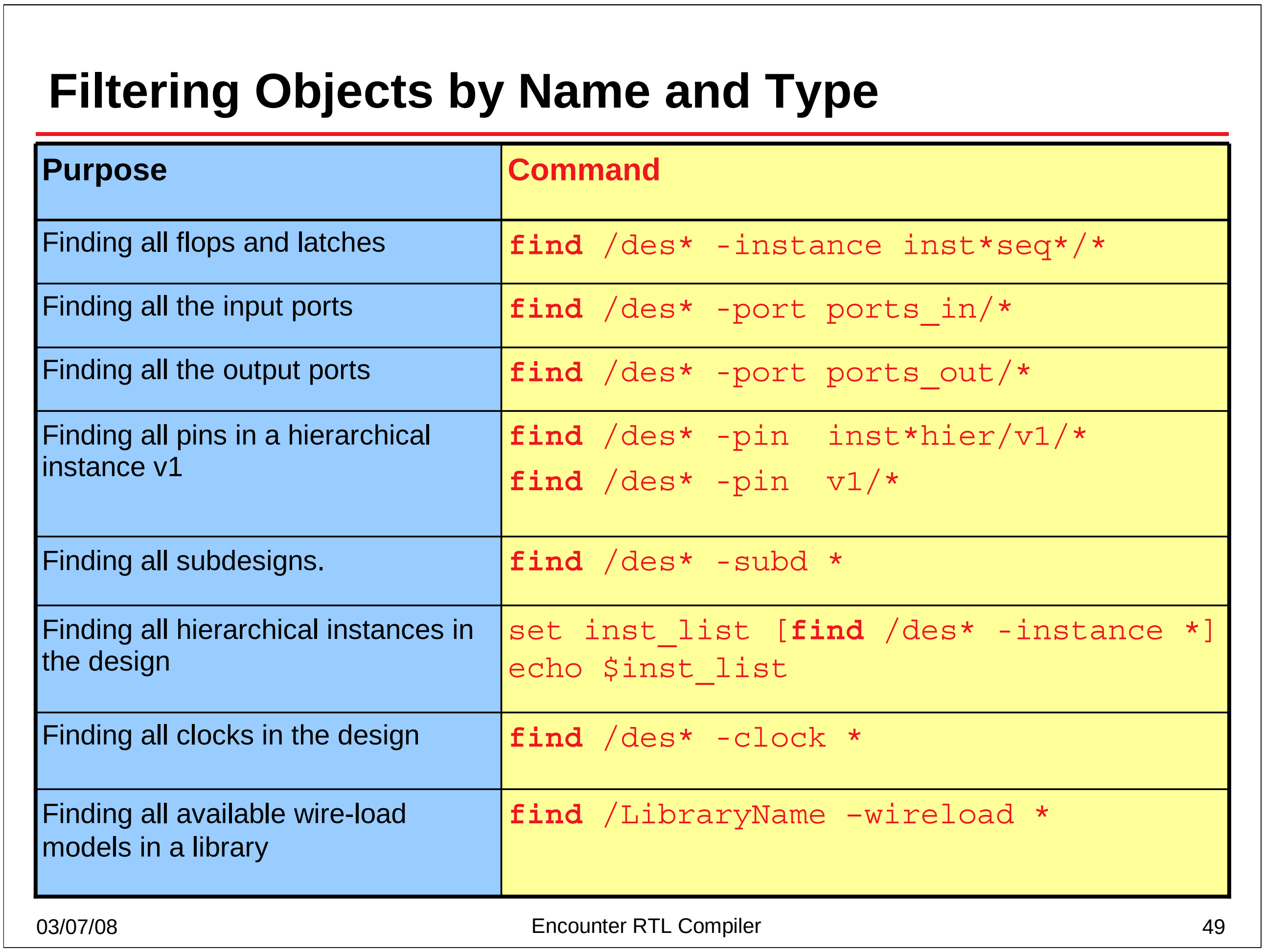
  `rm /des*/test3/timing/clock_domains/domain_1/clock`   Removes **clock** object.

- `find`: Finds an object and passes it to other commands.

  `ls -l -a [find / -wireload *]`   Reports the wire loads.

The software uses a database for its operations. Therefore, making modifications to the database will not alter the design files on the hard disk.

You need to manually save the modifications to the hard disk, by using the *write_hdl* command. Use a naming convention when writing files so that you do not overwrite any existing files.



Filtering Objects by Name and Type

Purpose	Command
Finding all flops and latches	`find /des* -instance inst*seq*/*`
Finding all the input ports	`find /des* -port ports_in/*`
Finding all the output ports	`find /des* -port ports_out/*`

Finding all pins in a hierarchical instance v1	`find /des* -pin  inst*hier/v1/*`
	`find /des* -pin  v1/*`

Finding all subdesigns.	`find /des* -subd *`

Finding all hierarchical instances in the design	`set inst_list [find /des* -instance *]`
	`echo $inst_list`

Finding all clocks in the design	`find /des* -clock *`
Finding all available wire-load models in a library	`find /LibraryName -wireload *`

03/07/08 Encounter RTL Compiler 49

The **find** command is the most convenient method to locate all objects that are part of the design hierarchy.

Syntax

```
find [<root_path>] [-maxdepth <integer>]
[-mindepth <integer>] [-ignorecase] [-vname]
[-option|*]+ <object>
```

To return all the available clocks in the design, enter:

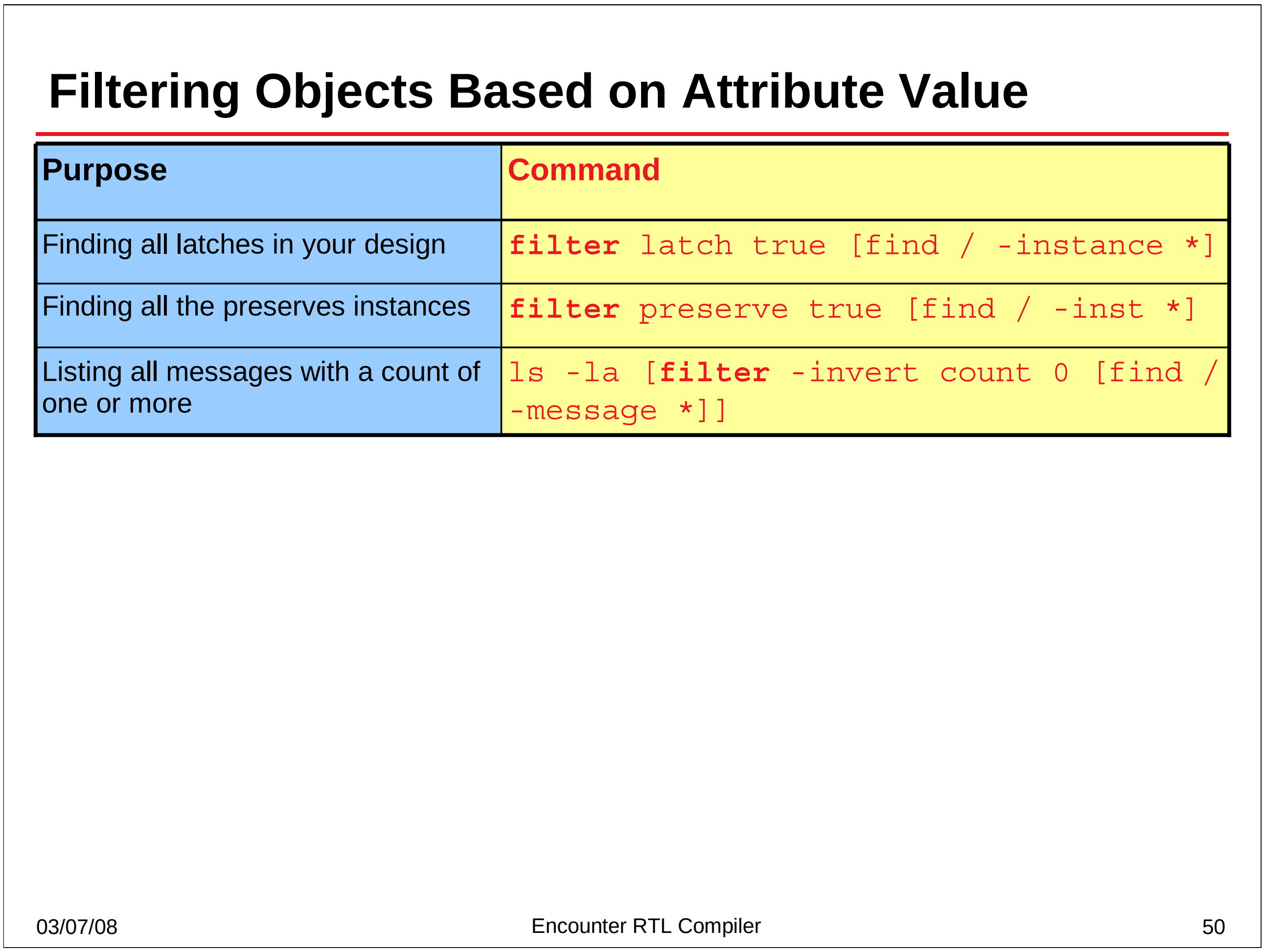
```
      find / -clock *
```

To return the name of the object after removing all the directory information, enter:

```
      basename [find / -clock *]
```

To return only the directory path of the object, enter:

```
      dirname [find / -clock *]
```



Filtering Objects Based on Attribute Value

Purpose	Command
Finding all latches in your design	**filter** latch true [find / -instance *]
Finding all the preserves instances	**filter** preserve true [find / -inst *]
Listing all messages with a count of one or more	ls -la [**filter** -invert count 0 [find / -message *]]

To filter objects with a certain attribute value, use this command:

```
filter [-invert] [-regexp] attr_name attr_value ...
  [object_list...]
```

Command Line Editing Keyboard Shortcuts

Control-a	Goes to the beginning of the line.
Control-c	Stops the current RTL Compiler (RC) process and if pressed twice exits RC.
Control-e	Goes to the end of the line.
Control-k	Deletes all text to the end of line.
Control-n	Goes to the next command in the history.
Control-p	Goes to the previous command in the history.
Control-z	Instructs RC to go to sleep.
Up arrow	Displays previous command in history.
Down arrow	Displays next command in history.
Left arrow	Moves the cursor right.
Right arrow	Moves the cursor left.
Backspace	Deletes the character to the left of the cursor.
Delete	Deletes the character to the right of the cursor.

Lab Exercises

Lab 2-1 Running the Basic Synthesis Flow

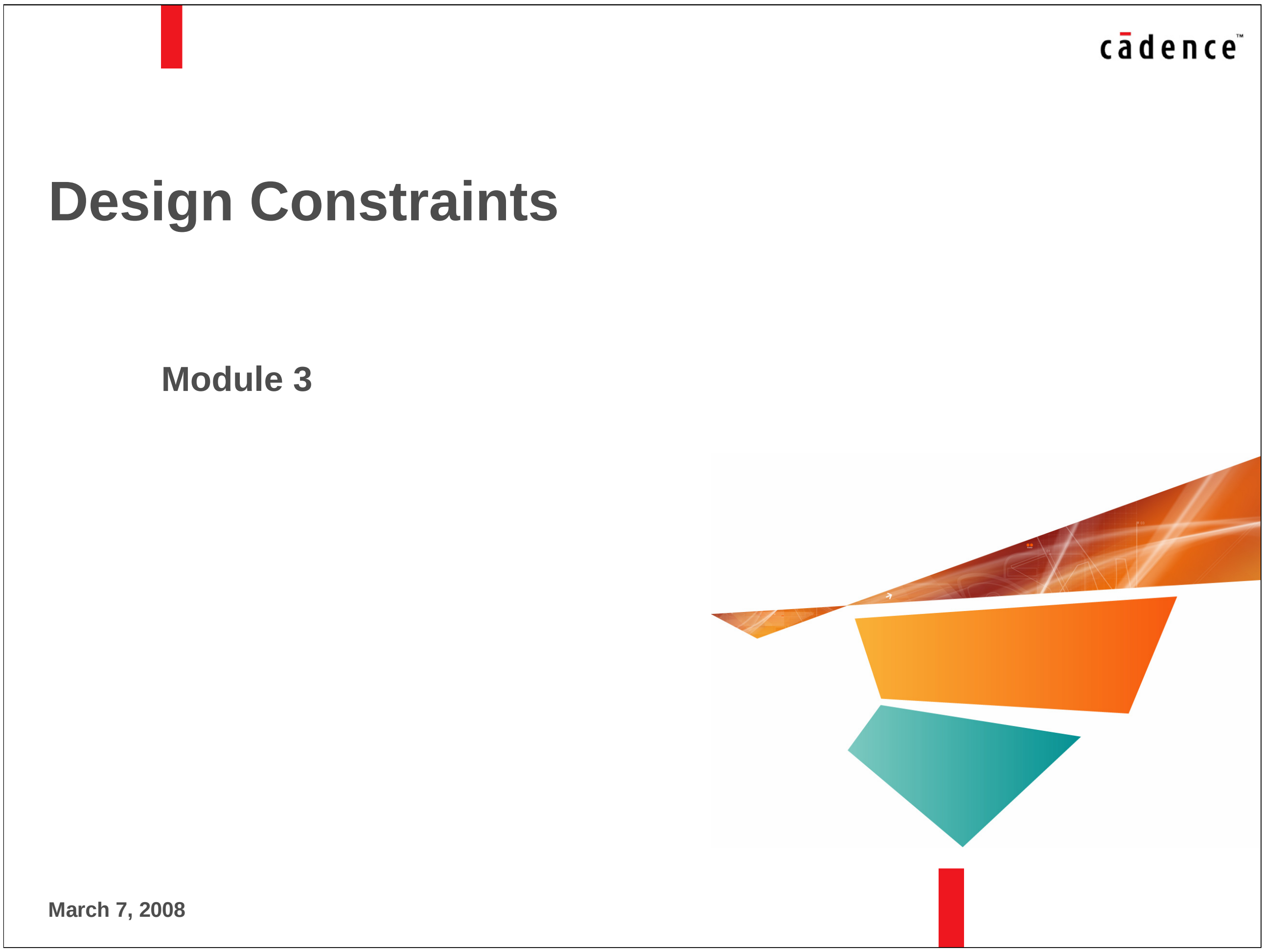
- Starting the Compiler
- Setting Up the Environment
- Reading Libraries
- Reading and Elaborating Designs
- Reading Constraints
- Synthesizing Your Design
- Writing Synthesis Outputs

Lab 2-2 Navigating the Design Hierarchy

- Filtering Objects by Type and Name
- Filtering Objects Based on Attribute Value
- Using Procedures

Lab 2-3 Using the Graphical Interface

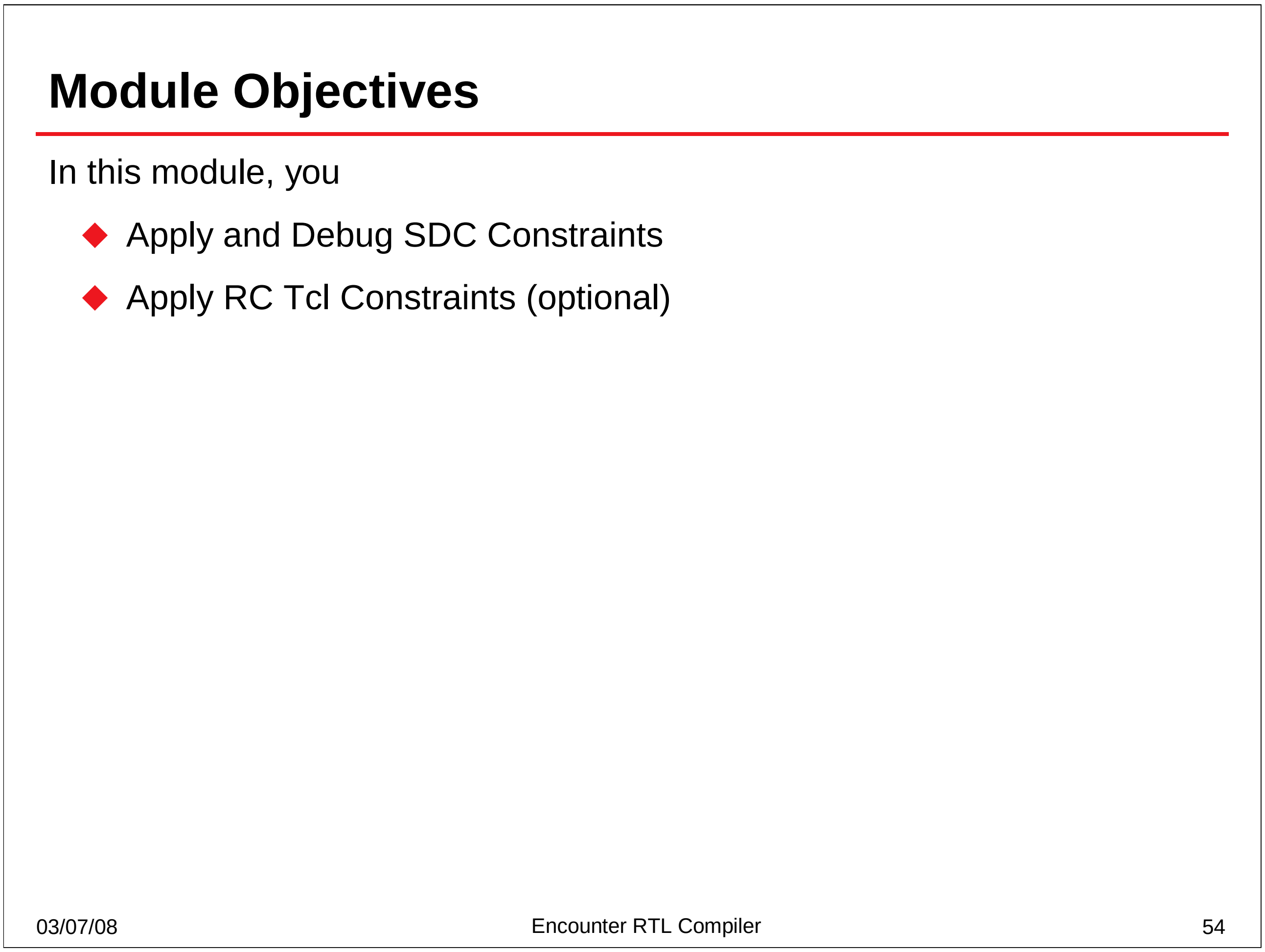
- Viewing Logical Hierarchy, HDL, and Schematic
- Setting Preferences
- Using the Tools Menu
- Generating Reports



Design Constraints

Module 3

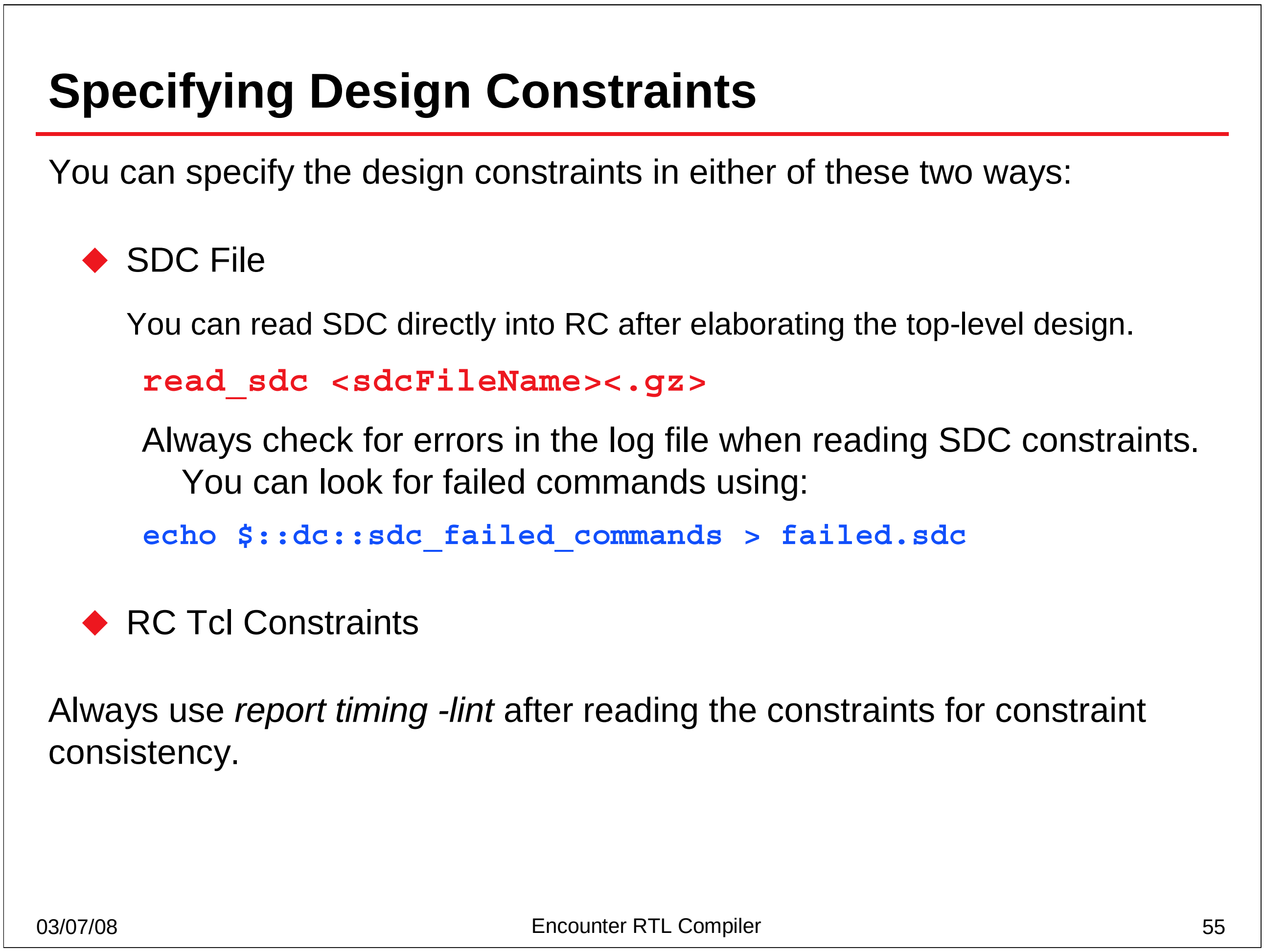
March 7, 2008



Module Objectives

In this module, you

- Apply and Debug SDC Constraints
- Apply RC Tcl Constraints (optional)



Specifying Design Constraints

You can specify the design constraints in either of these two ways:

- SDC File

  You can read SDC directly into RC after elaborating the top-level design.

  ```
  read_sdc <sdcFileName><.gz>
  ```

  Always check for errors in the log file when reading SDC constraints. You can look for failed commands using:

  ```
  echo $::dc::sdc_failed_commands > failed.sdc
  ```

- RC Tcl Constraints

Always use *report timing -lint* after reading the constraints for constraint consistency.

03/07/08 Encounter RTL Compiler 55

Use an SDC file when you import a design from other synthesis environments like BuildGates® Extreme or Design Compiler.

When using an SDC file, the capacitance specified in picofarads *(pF)* (SDC unit) is converted to femtofarads *(fF)* (RC unit) and time specified in *ns* is converted to *ps*.

You can use SDC commands interactively by using the **dc::** as a prefix. But when mixing DC commands and Encounter® RTL Compiler (RC) commands, be very careful with the units of capacitance and delay.

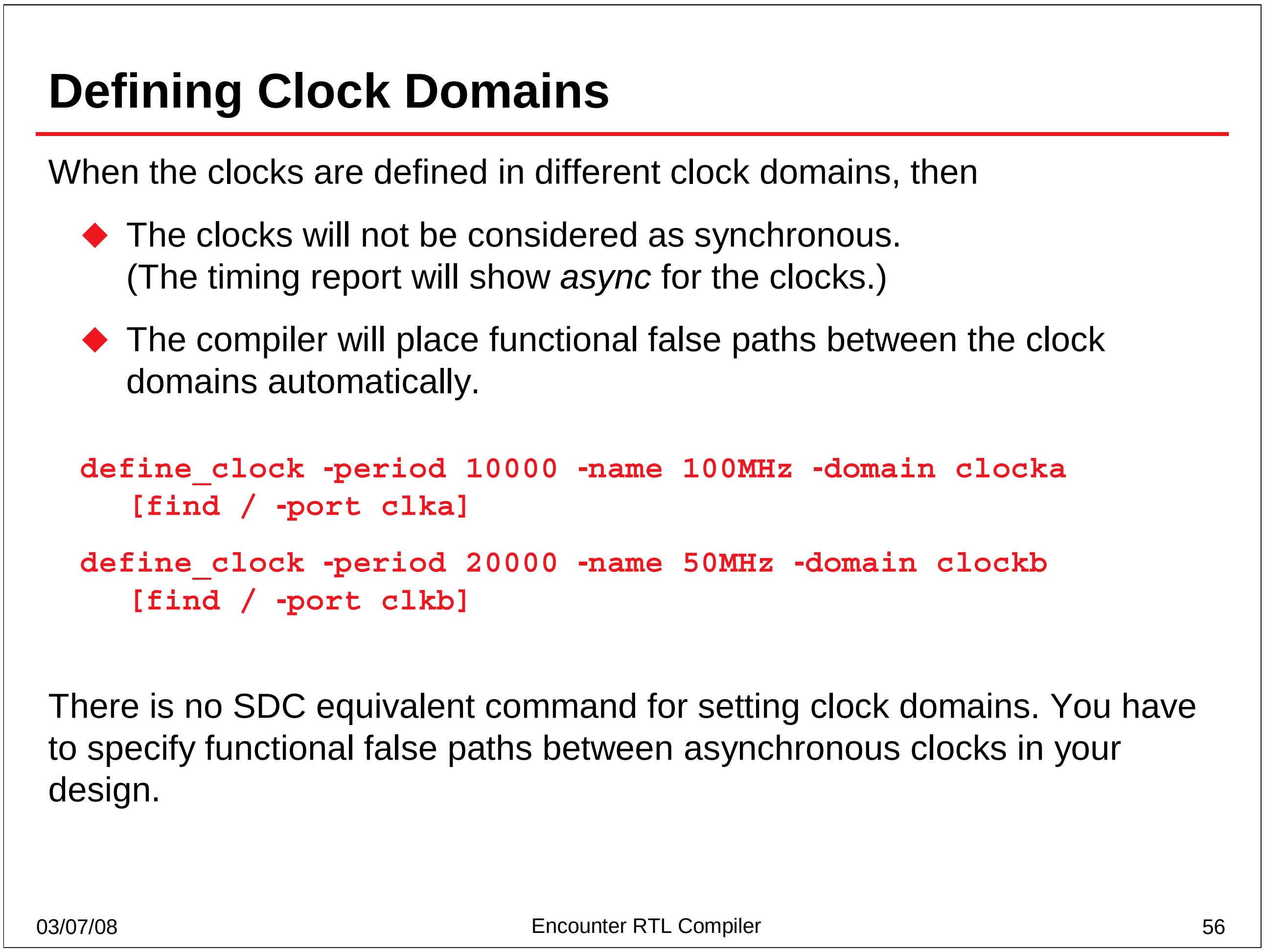
```
dc::set_load [get_attr load slow/INVX1/A] [dc::all_outputs]
```

In this case, the capacitance on all outputs will be off by a factor of 1000 because of the conversion from pF to fF. For example, *get_attr* , which is an RC command returns a load value of *10000 fF* from INVX1/A. The DC command *set_load*, is expecting loads in DC units (*pF*), and sets a load of *10000 pF* on all the outputs.

Instead, use separate commands with conversion factor included.

```
set loadIV [ expr [get_attr load slow/INVX1/A]/1000]
dc:: set_load loadIV [dc::all_outputs]
```

Remember to use **dc::** with every command, even if used recursively (command within a command).



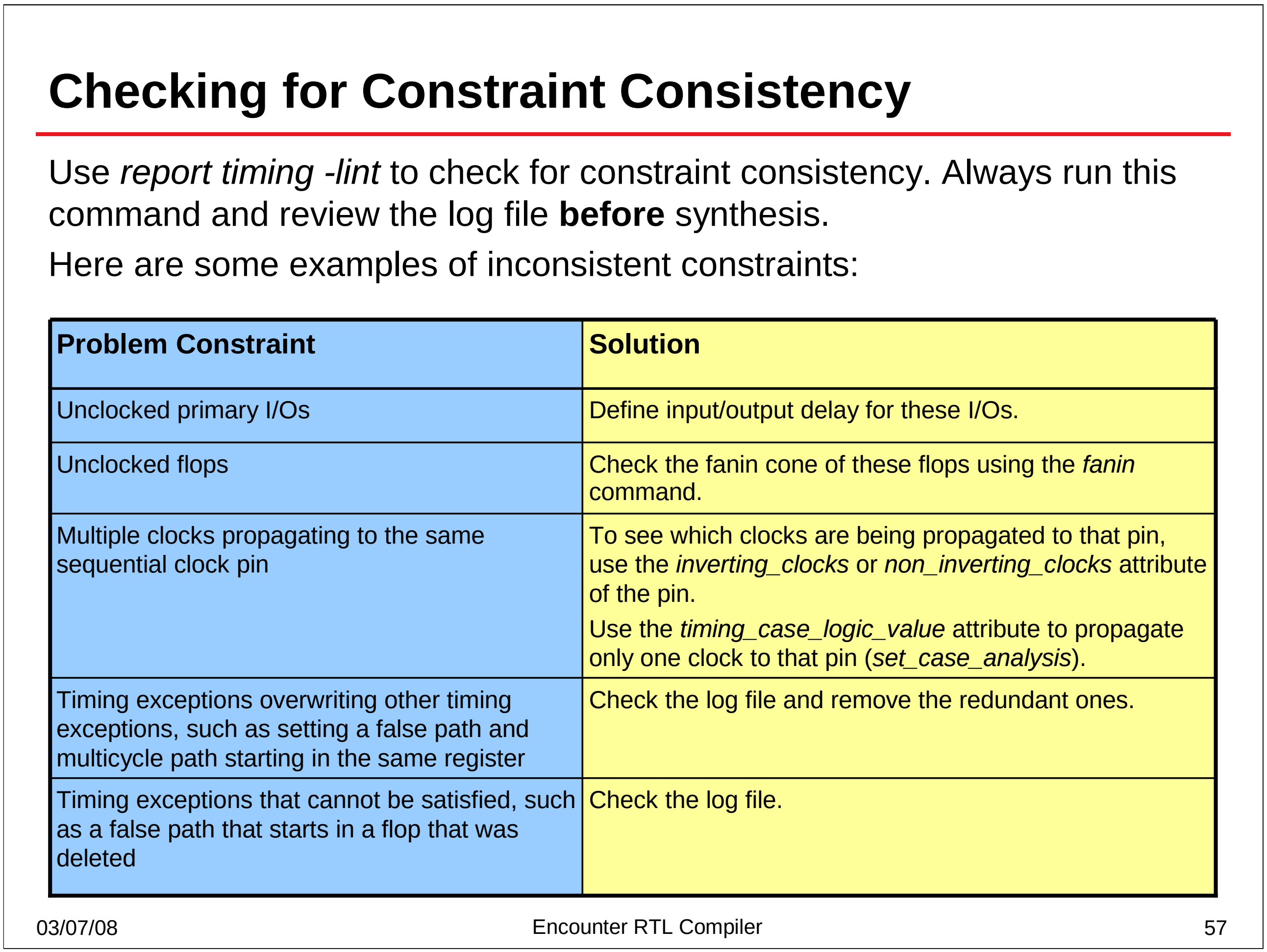
Defining Clock Domains

When the clocks are defined in different clock domains, then

- The clocks will not be considered as synchronous.
  (The timing report will show *async* for the clocks.)
- The compiler will place functional false paths between the clock domains automatically.

```
define_clock -period 10000 -name 100MHz -domain clocka
   [find / -port clka]
define_clock -period 20000 -name 50MHz -domain clockb
   [find / -port clkb]
```

There is no SDC equivalent command for setting clock domains. You have to specify functional false paths between asynchronous clocks in your design.



Checking for Constraint Consistency

Use *report timing -lint* to check for constraint consistency. Always run this command and review the log file **before** synthesis.

Here are some examples of inconsistent constraints:

Problem Constraint	**Solution**
Unclocked primary I/Os	Define input/output delay for these I/Os.
Unclocked flops	Check the fanin cone of these flops using the *fanin* command.
Multiple clocks propagating to the same sequential clock pin	To see which clocks are being propagated to that pin, use the *inverting_clocks* or *non_inverting_clocks* attribute of the pin. Use the *timing_case_logic_value* attribute to propagate only one clock to that pin (*set_case_analysis*).
Timing exceptions overwriting other timing exceptions, such as setting a false path and multicycle path starting in the same register	Check the log file and remove the redundant ones.
Timing exceptions that cannot be satisfied, such as a false path that starts in a flop that was deleted	Check the log file.

03/07/08 Encounter RTL Compiler 57

Use *report timing -lint* after elaborating the design and reading the SDC file to generate a detailed timing problem report.

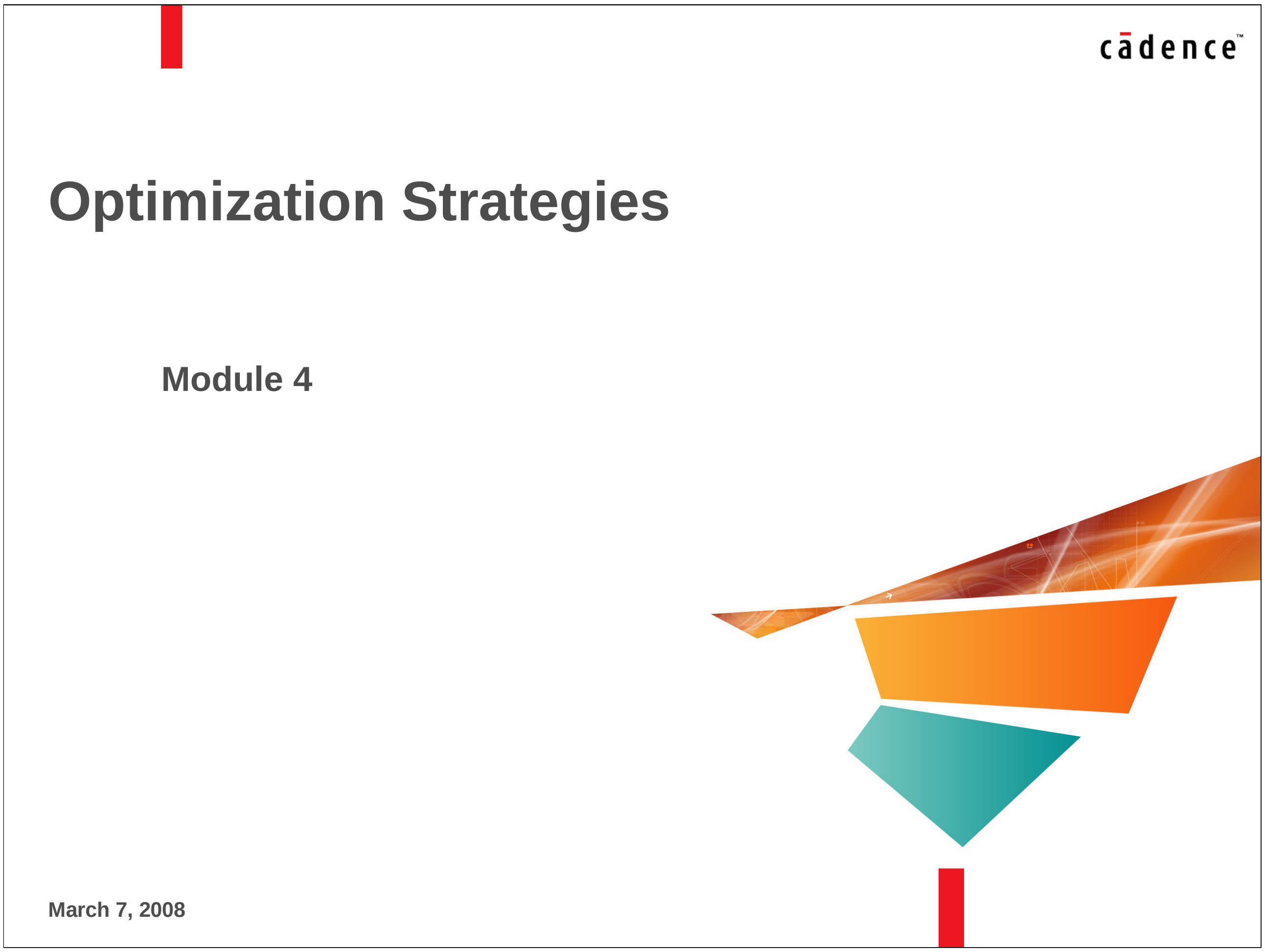
Categories of constraint issues reported

- Conflicting or unresolved exceptions
- Combinational loops
- Conflicting case constraints
- Missing external delays
- Missing or undefined clocks
- Multiple drivers
- Multimode constraint issues

Lab Exercises

Lab 3-1 Reading SDC Constraints

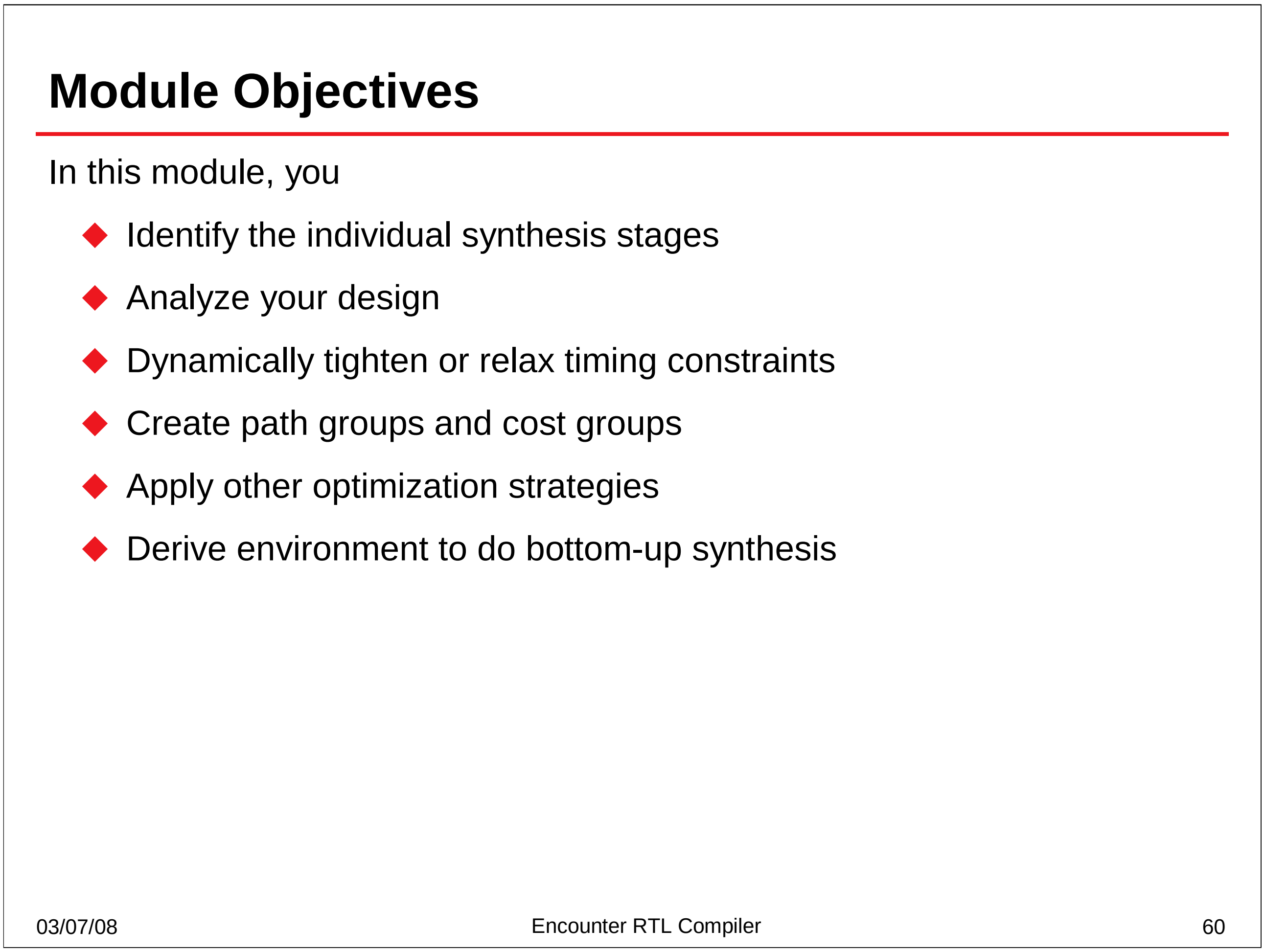
- ❍ Setting Up the Environment
- ❍ Debugging Failed SDC Constraints
- ❍ Analyzing Missing SDC Constraints



Optimization Strategies

Module 4

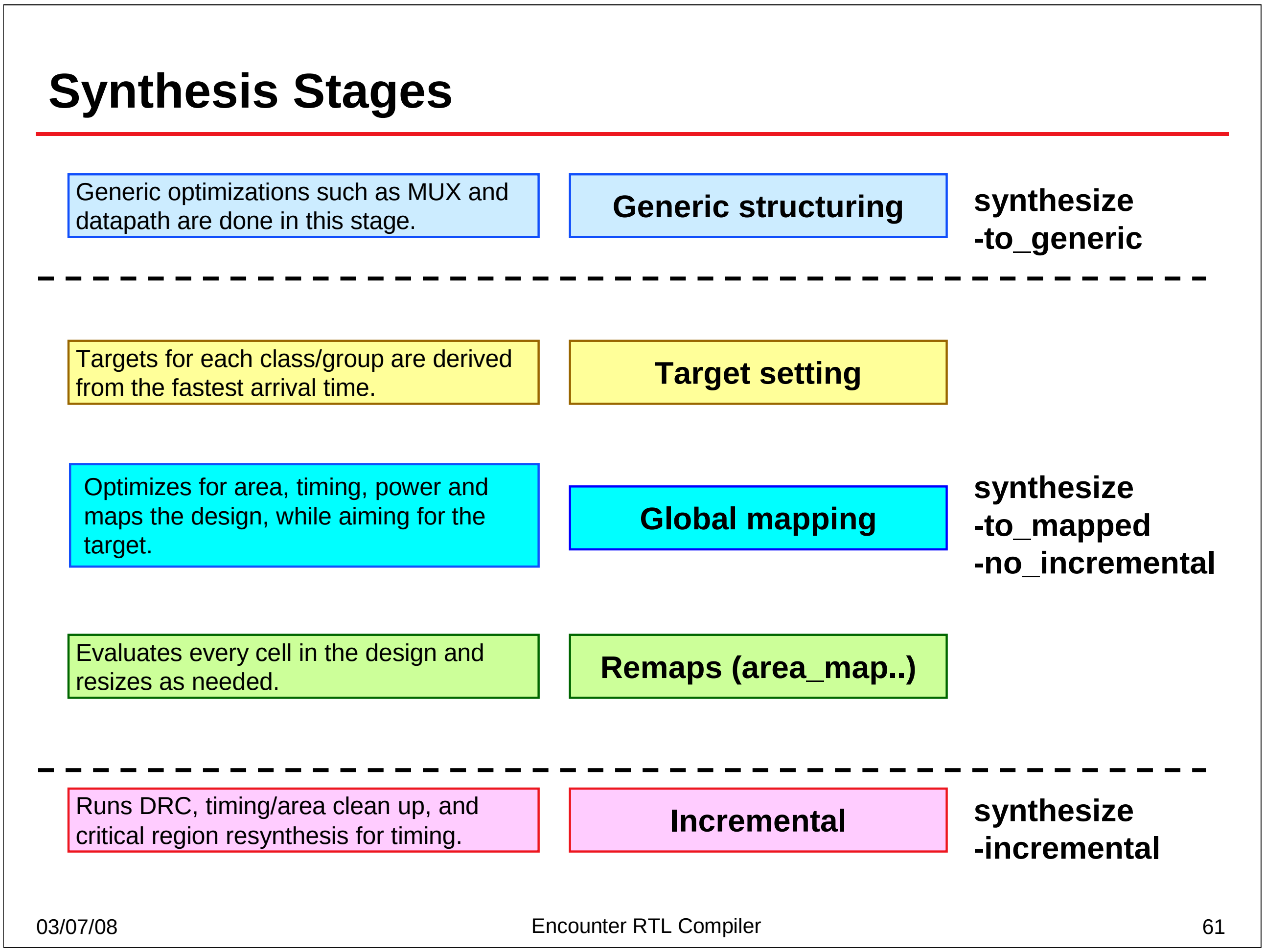
March 7, 2008



Module Objectives

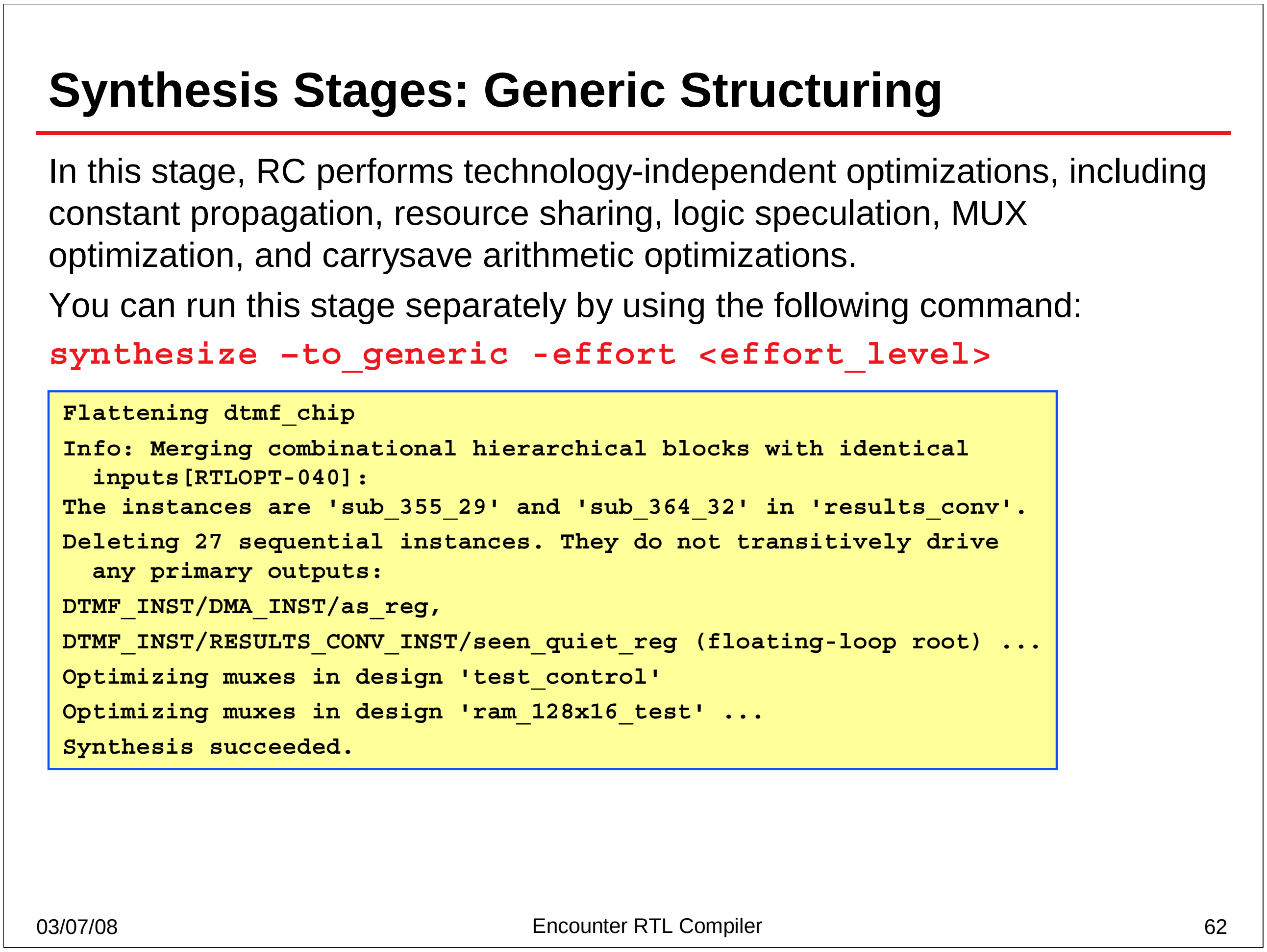
In this module, you

- Identify the individual synthesis stages
- Analyze your design
- Dynamically tighten or relax timing constraints
- Create path groups and cost groups
- Apply other optimization strategies
- Derive environment to do bottom-up synthesis



Synthesis Stages

Generic optimizations such as MUX and datapath are done in this stage.	**Generic structuring**	**synthesize -to_generic**
Targets for each class/group are derived from the fastest arrival time.	**Target setting**	**synthesize -to_mapped -no_incremental**
Optimizes for area, timing, power and maps the design, while aiming for the target.	**Global mapping**	
Evaluates every cell in the design and resizes as needed.	**Remaps (area_map..)**	
Runs DRC, timing/area clean up, and critical region resynthesis for timing.	**Incremental**	**synthesize -incremental**



Synthesis Stages: Generic Structuring

In this stage, RC performs technology-independent optimizations, including constant propagation, resource sharing, logic speculation, MUX optimization, and carrysave arithmetic optimizations.

You can run this stage separately by using the following command:

```
synthesize -to_generic -effort <effort_level>
```

```
Flattening dtmf_chip
Info: Merging combinational hierarchical blocks with identical
  inputs[RTLOPT-040]:
The instances are 'sub_355_29' and 'sub_364_32' in 'results_conv'.
Deleting 27 sequential instances. They do not transitively drive
  any primary outputs:
DTMF_INST/DMA_INST/as_reg,
DTMF_INST/RESULTS_CONV_INST/seen_quiet_reg (floating-loop root) ...
Optimizing muxes in design 'test_control'
Optimizing muxes in design 'ram_128x16_test' ...
Synthesis succeeded.
```

The *synthesize -to_generic* command does RTL optimization. This is automatically done within *synthesize -to_mapped* using *medium* effort, if your design comes from RTL. When loading a Verilog® netlist, do not use this command unless it is absolutely necessary, because it will unmap the design.

The *medium* effort is the default choice. You can use *high* effort for datapath intensive designs, or designs for which it is hard to meet timing.

The compiler optimizes sequential instances that transitively do not fan out to primary output. This information is reported in the log file. If you see unmapped points in formal verification, check for deleted sequential instances in the log file.

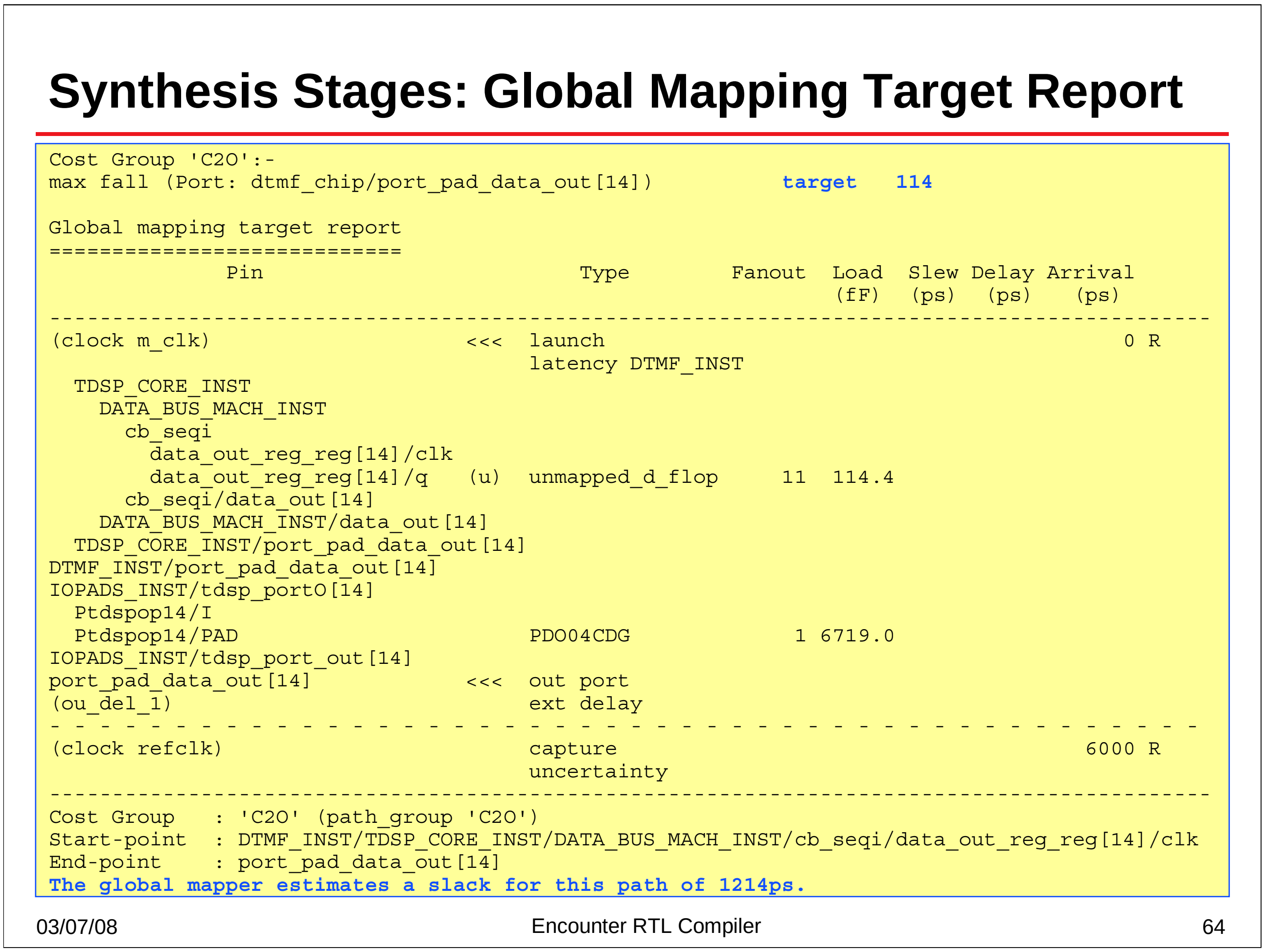
When you use this command on a mapped netlist, it unmaps the design.

Synthesis Stages: Pretarget Global Mapping

In this first phase of global mapping, RC performs tentative structuring and computes the estimated arrival and required times for all the endpoints, based on the effort level that you set. The result of this stage is the target for each cost group.

synthesize -to_mapped -effort <effort_level>

```
Mapping dtmf_chip to gates.
@Subdesign G2C_DP_subdec has 26 instances - the first is
  DTMF_INST/RESULTS_CONV_INST/sub_332_21
@Unfolding 'G2C_DP_addinc3408' instances due to different
  environments: different TBR inputs;
splitting 'DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/add_1352_31'
  and 2 other instances
Structuring (delay-based) logic partition in alu_32...
Performing redundancy-removal...
Performing bdd-opto...
Performing redundancy-removal...
Done structuring (delay-based) logic partition in alu_32
Mapping logic partition in alu_32...
```

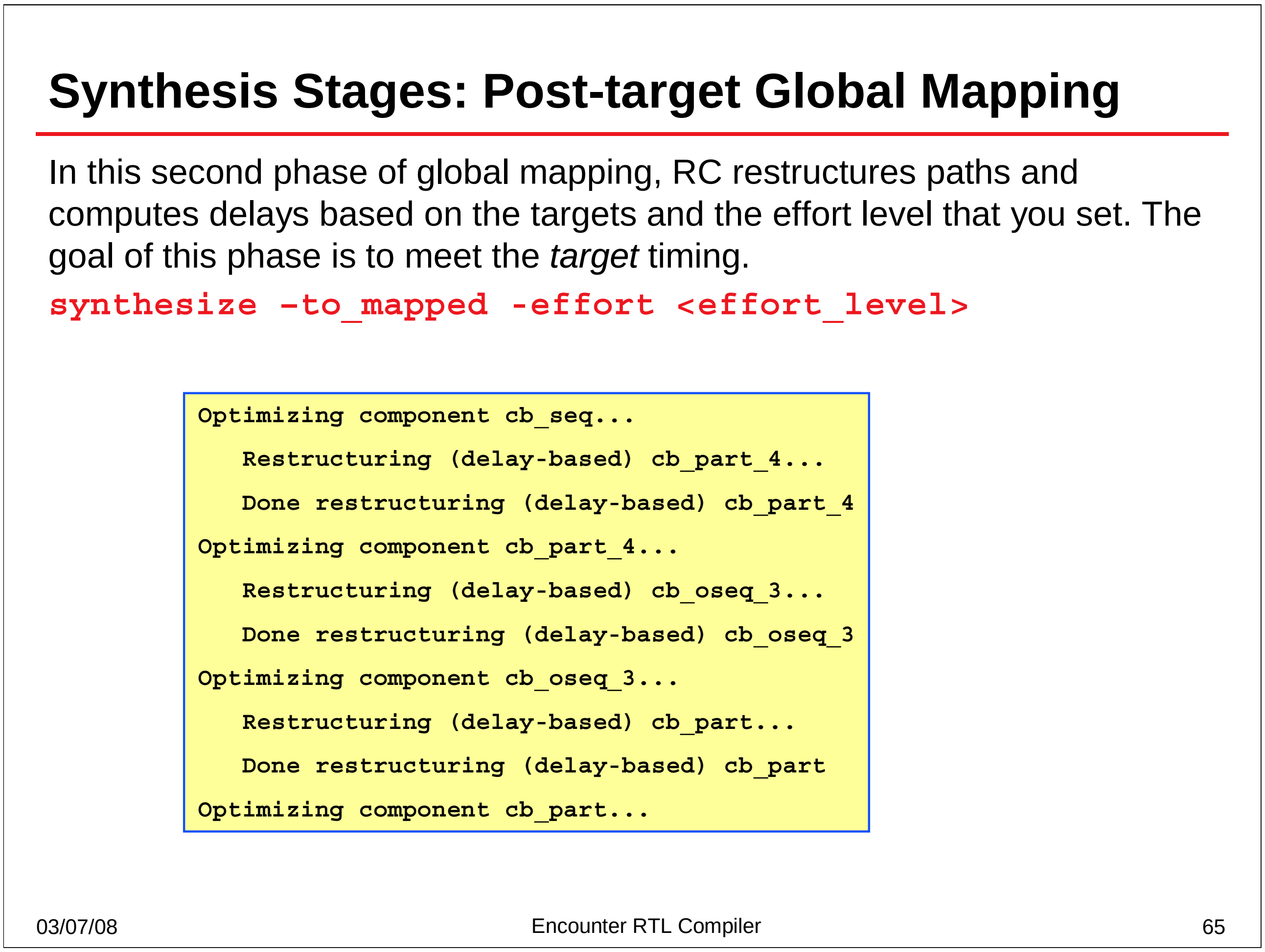


Synthesis Stages: Global Mapping Target Report

```
Cost Group 'C2O':-
max fall (Port: dtmf_chip/port_pad_data_out[14])          target   114

Global mapping target report
============================
              Pin                         Type        Fanout  Load  Slew Delay Arrival
                                                              (fF)  (ps)  (ps)    (ps)
--------------------------------------------------------------------------------------------
(clock m_clk)                    <<<  launch                                             0  R
                                      latency DTMF_INST
  TDSP_CORE_INST
    DATA_BUS_MACH_INST
      cb_seqi
        data_out_reg_reg[14]/clk
        data_out_reg_reg[14]/q   (u)  unmapped_d_flop     11  114.4
      cb_seqi/data_out[14]
    DATA_BUS_MACH_INST/data_out[14]
  TDSP_CORE_INST/port_pad_data_out[14]
DTMF_INST/port_pad_data_out[14]
IOPADS_INST/tdsp_portO[14]
  Ptdspop14/I
  Ptdspop14/PAD                       PDO04CDG             1 6719.0
IOPADS_INST/tdsp_port_out[14]
port_pad_data_out[14]            <<<  out port
(ou_del_1)                            ext delay
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock refclk)                        capture                                         6000  R
                                      uncertainty
--------------------------------------------------------------------------------------------
Cost Group   : 'C2O' (path_group 'C2O')
Start-point  : DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_INST/cb_seqi/data_out_reg_reg[14]/clk
End-point    : port_pad_data_out[14]
The global mapper estimates a slack for this path of 1214ps.
```

The worst case slack report with a target slack is written during the global mapping stage by entering:

```
set_attribute map_timing true /
```



Synthesis Stages: Post-target Global Mapping

In this second phase of global mapping, RC restructures paths and computes delays based on the targets and the effort level that you set. The goal of this phase is to meet the *target* timing.

```
synthesize -to_mapped -effort <effort_level>
```

```
Optimizing component cb_seq...
   Restructuring (delay-based) cb_part_4...
   Done restructuring (delay-based) cb_part_4
Optimizing component cb_part_4...
   Restructuring (delay-based) cb_oseq_3...
   Done restructuring (delay-based) cb_oseq_3
Optimizing component cb_oseq_3...
   Restructuring (delay-based) cb_part...
   Done restructuring (delay-based) cb_part
Optimizing component cb_part...
```

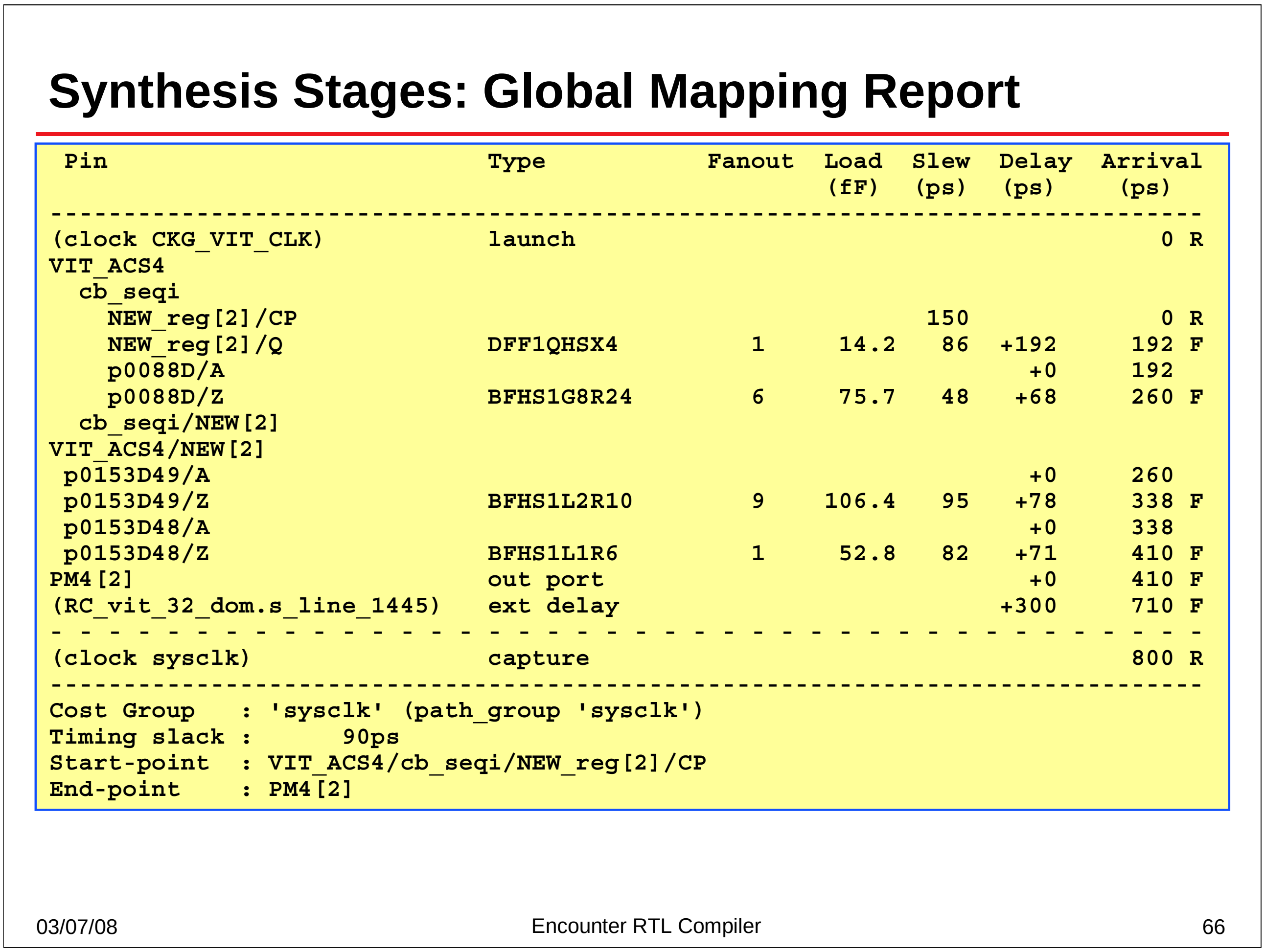
Effort low: The design is mapped to gates, but with this option, the Encounter® RTL Compiler does very little RTL optimization, incremental clean up, or redundancy identification and removal. The low setting is generally not recommended.

Effort medium (default setting): RTL Compiler performs better timing-driven structuring, incremental synthesis, and redundancy identification and removal on the design.

Effort high: RTL Compiler does the timing-driven structuring on larger sections of logic and spends more time and makes more attempts on incremental clean up. This effort level involves very aggressive redundancy identification and removal.

Unfolding means to uniquely instantiate the blocks, based on their environment.

You can generate the area reports in different stages during synthesis. However, you need to focus on the area report after all the synthesis phases finish. The final area numbers will be similar to what is generated during the incremental mapping stage under final optimization status.



Synthesis Stages: Global Mapping Report

```
 Pin                           Type            Fanout  Load  Slew  Delay  Arrival
                                                       (fF)  (ps)  (ps)    (ps)
--------------------------------------------------------------------------------
(clock CKG_VIT_CLK)            launch                                        0 R
VIT_ACS4
  cb_seqi
    NEW_reg[2]/CP                                             150            0 R
    NEW_reg[2]/Q               DFF1QHSX4            1   14.2   86   +192   192 F
    p0088D/A                                                          +0   192
    p0088D/Z                   BFHS1G8R24           6   75.7   48    +68   260 F
  cb_seqi/NEW[2]
VIT_ACS4/NEW[2]
 p0153D49/A                                                           +0   260
 p0153D49/Z                    BFHS1L2R10           9  106.4   95    +78   338 F
 p0153D48/A                                                           +0   338
 p0153D48/Z                    BFHS1L1R6            1   52.8   82    +71   410 F
PM4[2]                         out port                               +0   410 F
(RC_vit_32_dom.s_line_1445)    ext delay                            +300   710 F
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock sysclk)                 capture                                     800 R
--------------------------------------------------------------------------------
Cost Group    : 'sysclk' (path_group 'sysclk')
Timing slack :       90ps
Start-point   : VIT_ACS4/cb_seqi/NEW_reg[2]/CP
End-point     : PM4[2]
```

The worst case slack report with a target slack is written during the global mapping stage by entering:

```
set_attribute map_timing true /
```

When you use the above attribute in conjunction with the following variable, the compiler gives the targets and fancy naming styles that can help debug your design.

```
set map_fancy_names 1
```

The compiler uses slack and optimization status to name the cells in the critical path of each cost group.

Synthesis Stages: Remapping

Several optimization routines are used during this stage of synthesis mainly to reduce the area of the design.

```
Global mapping status
=====================
                         Group
                         Total
                 Total   Worst
Operation         Area   Slacks Worst Path
-------------------------------------------------------------------------
 global_map     721782    -308  VIT_ACS10/NEW_reg[5]/CP -->
VIT_ACS26/NEW_reg[1]/D
 fine_map       514143    -372  VIT_ACS10/NEW_reg[6]/CP -->
VIT_ACS8/NEW_reg[2]/D
 area_map       512565    -344  VIT_ACS23/NEW_reg[4]/CP -->
VIT_ACS31/NEW_reg[7]/D
 area_map       500797    -346  VIT_ACS18/NEW_reg[7]/CP -->
VIT_ACS27/NEW_reg[6]/D
 area_map       498515    -345  VIT_ACS1/NEW_reg[5]/CP -->
VIT_ACS4/THREE_SELECT_REG_reg/D
Done mapping dtmf_chip
```

Remaps such as *area_map* and *fine_map* refine the timing and area of all critical paths by remapping the cells according to the new surroundings of the cell.

Synthesis Stages: Incremental Synthesis

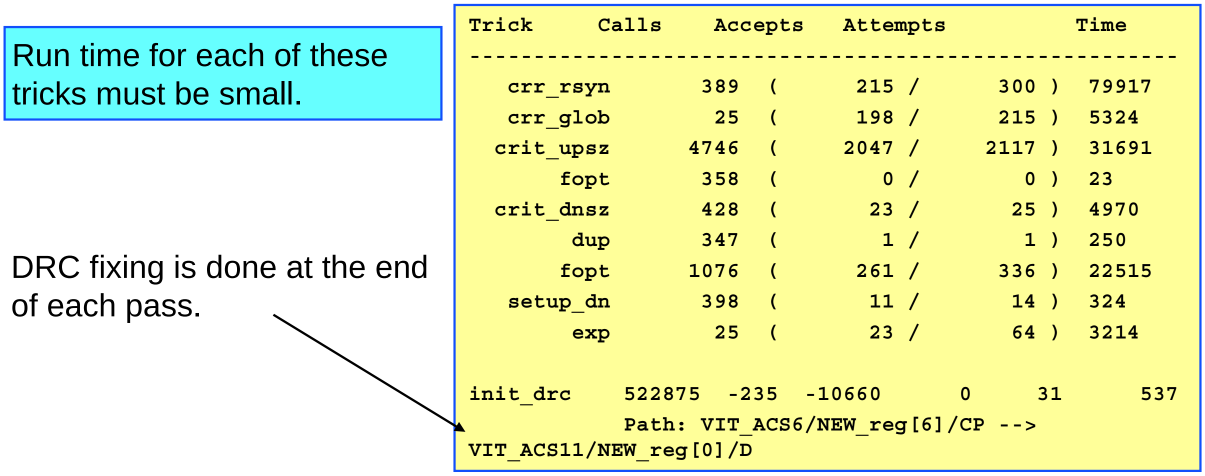
Run *synthesize -to_mapped* to achieve the timing goals of the design. To clean up timing using local optimizations, such as CRR, and to insert scan chains if enabled, run *synthesize -incremental*.

```
Incremental optimization status
===============================
                         Group
                         Total     Total  - - - - DRC Totals - - -
                 Total   Worst      Neg        Max       Max    Max
Operation         Area   Slacks     Slack      Trans       Cap  Fanout
-------------------------------------------------------------------
init_delay     498515    -345  -124671       414       18      229
            Path: VIT_ACS1/NEW_reg[5]/CP -->
VIT_ACS4/THREE_SELECT_REG_reg/D
 incr_delay     502638    -301  -114125       129       69      276
            Path: VIT_ACS16/NEW_reg[2]/CP --> VIT_ACS2/NEW_reg[6]/D
 incr_delay     511982    -267  -100144         0       19      614
            Path: VIT_ACS19/NEW_reg[6]/CP -->
VIT_ACS13/NEW_reg[1]/D
 incr_delay     515304    -221   -91064         0       34      614
            Path: VIT_ACS30/NEW_reg[2]/CP -->
VIT_ACS10/NEW_reg[0]/D
…
```

CRR: Critical region resynthesis

Synthesis Stages: Incremental Synthesis (continued)

This report shows the localized algorithms (tricks) used in incremental optimization, the corresponding number of attempts made by the synthesis engine, and the number of times that the routine has been run to improve the design goals.



Run time for each of these tricks must be small.

DRC fixing is done at the end of each pass.

```
Trick      Calls     Accepts    Attempts          Time
-------------------------------------------------------
   crr_rsyn        389   (       215 /       300 )  79917
   crr_glob         25   (       198 /       215 )  5324
  crit_upsz       4746   (      2047 /      2117 )  31691
       fopt        358   (         0 /         0 )  23
  crit_dnsz        428   (        23 /        25 )  4970
        dup        347   (         1 /         1 )  250
       fopt       1076   (       261 /       336 )  22515
   setup_dn        398   (        11 /        14 )  324
        exp         25   (        23 /        64 )  3214


init_drc     522875  -235  -10660      0     31       537
            Path: VIT_ACS6/NEW_reg[6]/CP -->
VIT_ACS11/NEW_reg[0]/D
```

To print the incremental optimization “tricks” in your report, you must set the *information_level* root attribute to 9.

```
set_attr information_level 9 /
```

As an alternative, you can use this command:

```
set iopt_stats 1
```

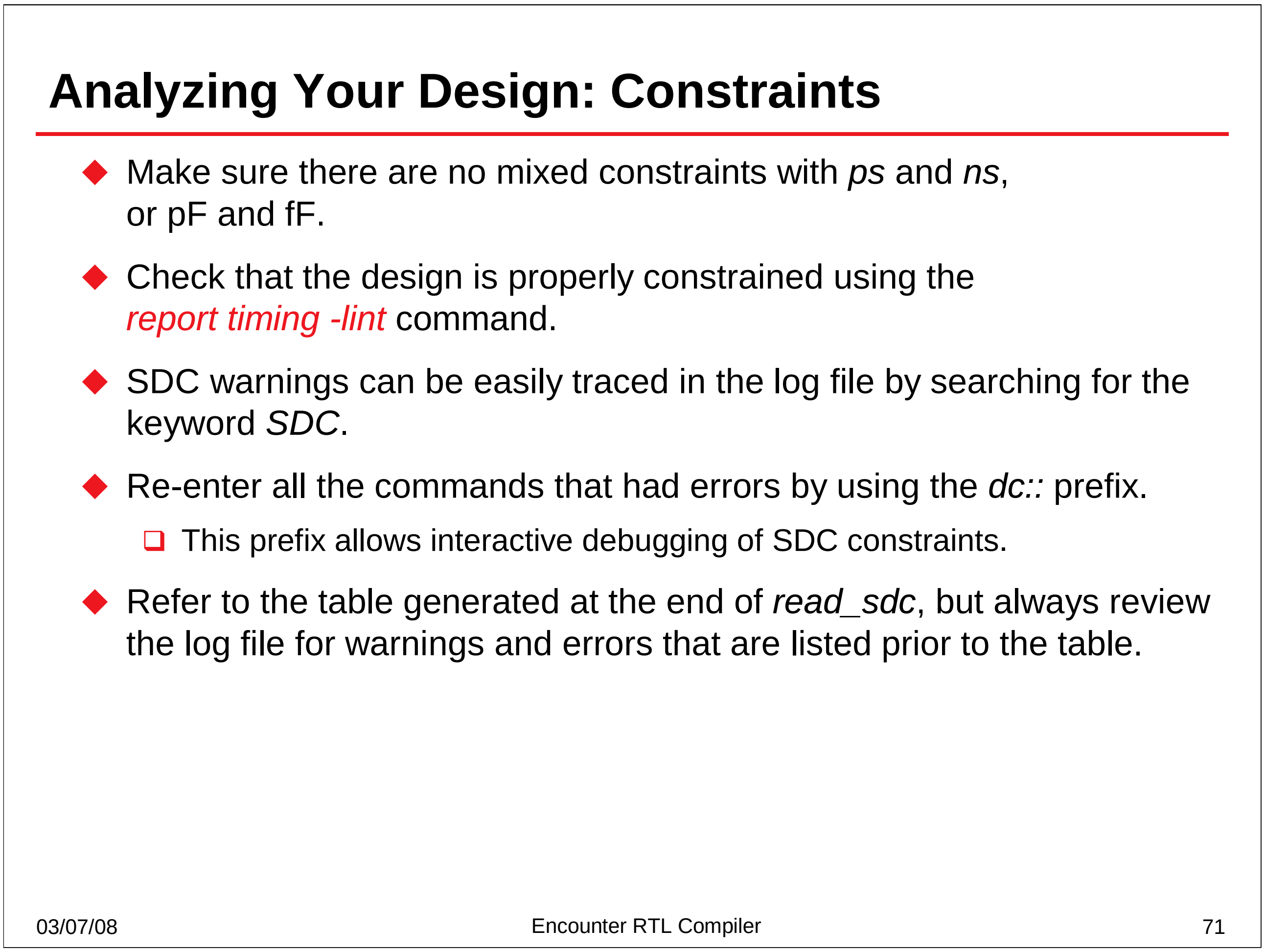
Analyzing Your Design: Elaborate

- After elaboration, check for the message '*Done elaborating*'.
- Check the logfile for any unresolved instances (The *unresolved* attribute will be set to *true* for an unresolved instance). Or use:
  `check_design -unresolved`
- Set the attribute *information_level* to 9 to get all the info messages.
- Make sure that there is only one top-level design.
- Review the log file for warnings and fix the issues. Synthesizing the design with elaboration issues can easily result in a bad netlist and poor QOR.

Common messages to pay attention to:

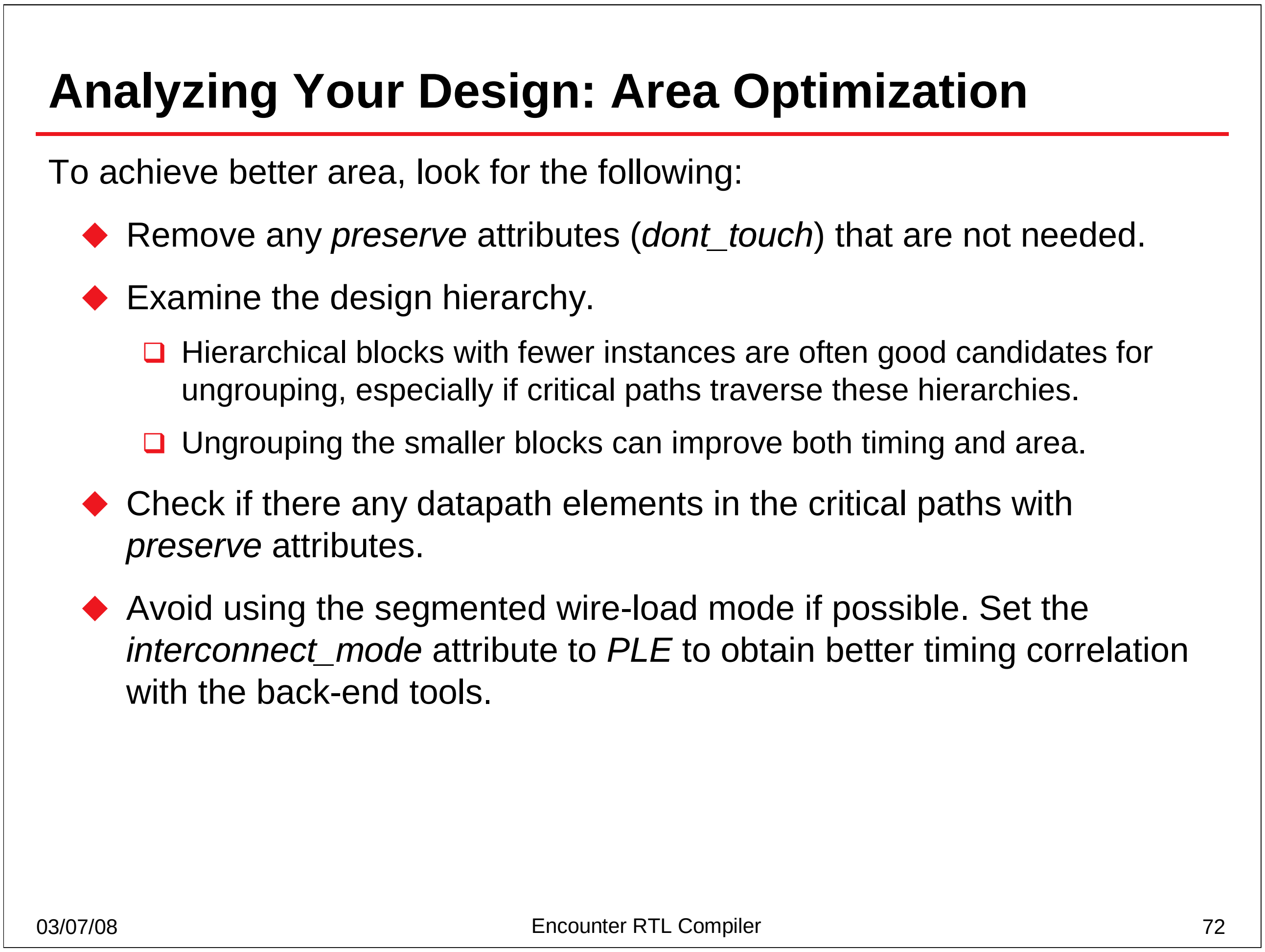
- Inconsistent nominal operating conditions (LBR-38)
- You cannot load a library that has no inverter or simple gate.
- Warning: Variable has multiple drivers. [ELAB-VLOG-14]

QOR: Quality of results



Analyzing Your Design: Constraints

- Make sure there are no mixed constraints with *ps* and *ns*, or pF and fF.
- Check that the design is properly constrained using the *report timing -lint* command.
- SDC warnings can be easily traced in the log file by searching for the keyword *SDC*.
- Re-enter all the commands that had errors by using the *dc::* prefix.
  - This prefix allows interactive debugging of SDC constraints.
- Refer to the table generated at the end of *read_sdc*, but always review the log file for warnings and errors that are listed prior to the table.

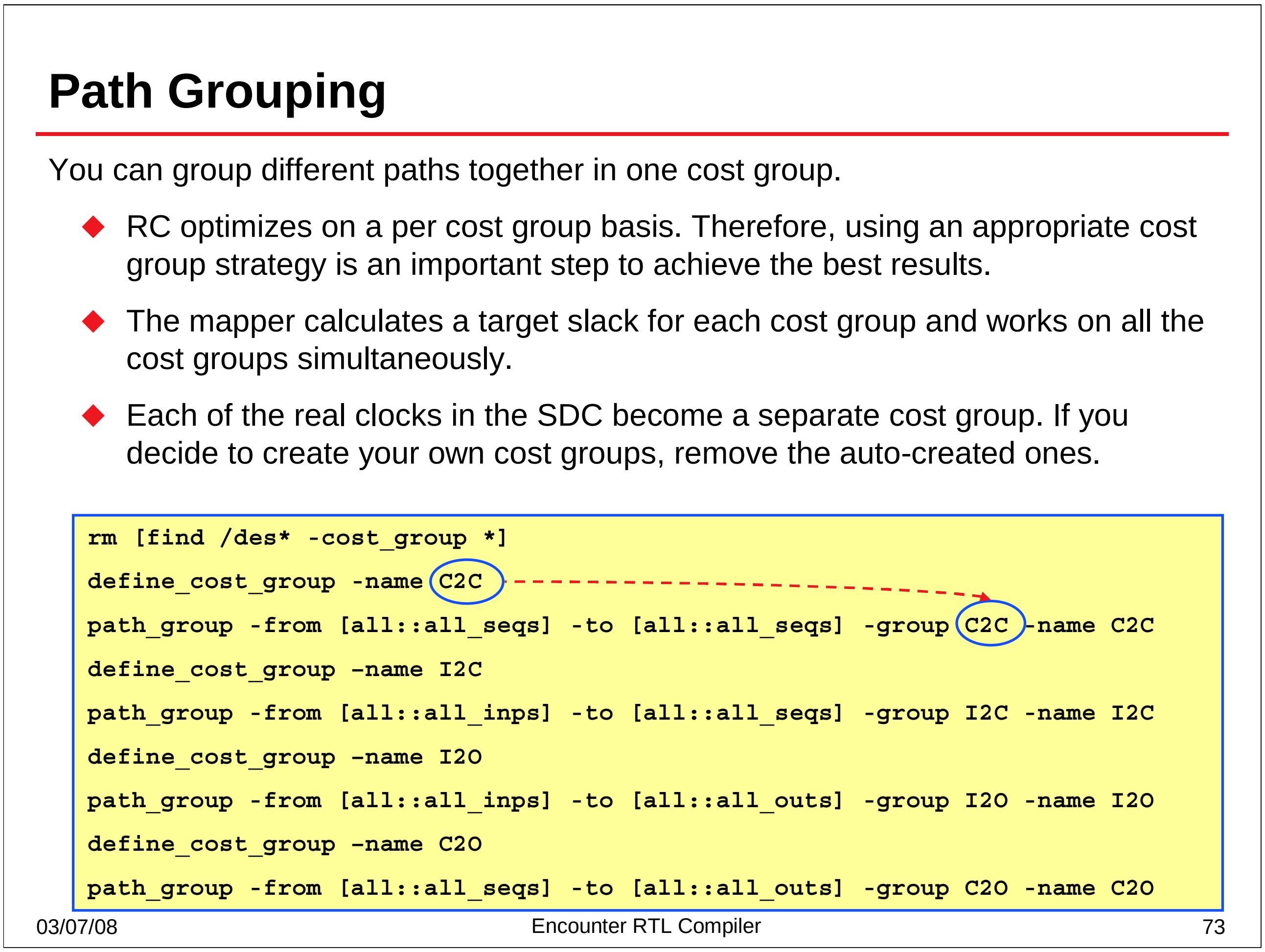


Analyzing Your Design: Area Optimization

To achieve better area, look for the following:

- Remove any *preserve* attributes (*dont_touch*) that are not needed.
- Examine the design hierarchy.
  - Hierarchical blocks with fewer instances are often good candidates for ungrouping, especially if critical paths traverse these hierarchies.
  - Ungrouping the smaller blocks can improve both timing and area.
- Check if there any datapath elements in the critical paths with *preserve* attributes.
- Avoid using the segmented wire-load mode if possible. Set the *interconnect_mode* attribute to *PLE* to obtain better timing correlation with the back-end tools.

PLE: Physical Layout Estimation



Path Grouping

You can group different paths together in one cost group.

- RC optimizes on a per cost group basis. Therefore, using an appropriate cost group strategy is an important step to achieve the best results.
- The mapper calculates a target slack for each cost group and works on all the cost groups simultaneously.
- Each of the real clocks in the SDC become a separate cost group. If you decide to create your own cost groups, remove the auto-created ones.

```
rm [find /des* -cost_group *]
define_cost_group -name C2C
path_group -from [all::all_seqs] -to [all::all_seqs] -group C2C -name C2C
define_cost_group -name I2C
path_group -from [all::all_inps] -to [all::all_seqs] -group I2C -name I2C
define_cost_group -name I2O
path_group -from [all::all_inps] -to [all::all_outs] -group I2O -name I2O
define_cost_group -name C2O
path_group -from [all::all_seqs] -to [all::all_outs] -group C2O -name C2O
```

03/07/08 Encounter RTL Compiler 73

To get the best performance goals from synthesis, you might need to apply optimization strategies, such as creating custom cost groups for paths in the design to change the synthesis cost function.

The following are the typical path groups:

Input-to-Output paths (I2O)

Input-to-Register paths (I2C)

Register-to-Register (C2C)

Register-to-Output paths (C2O)

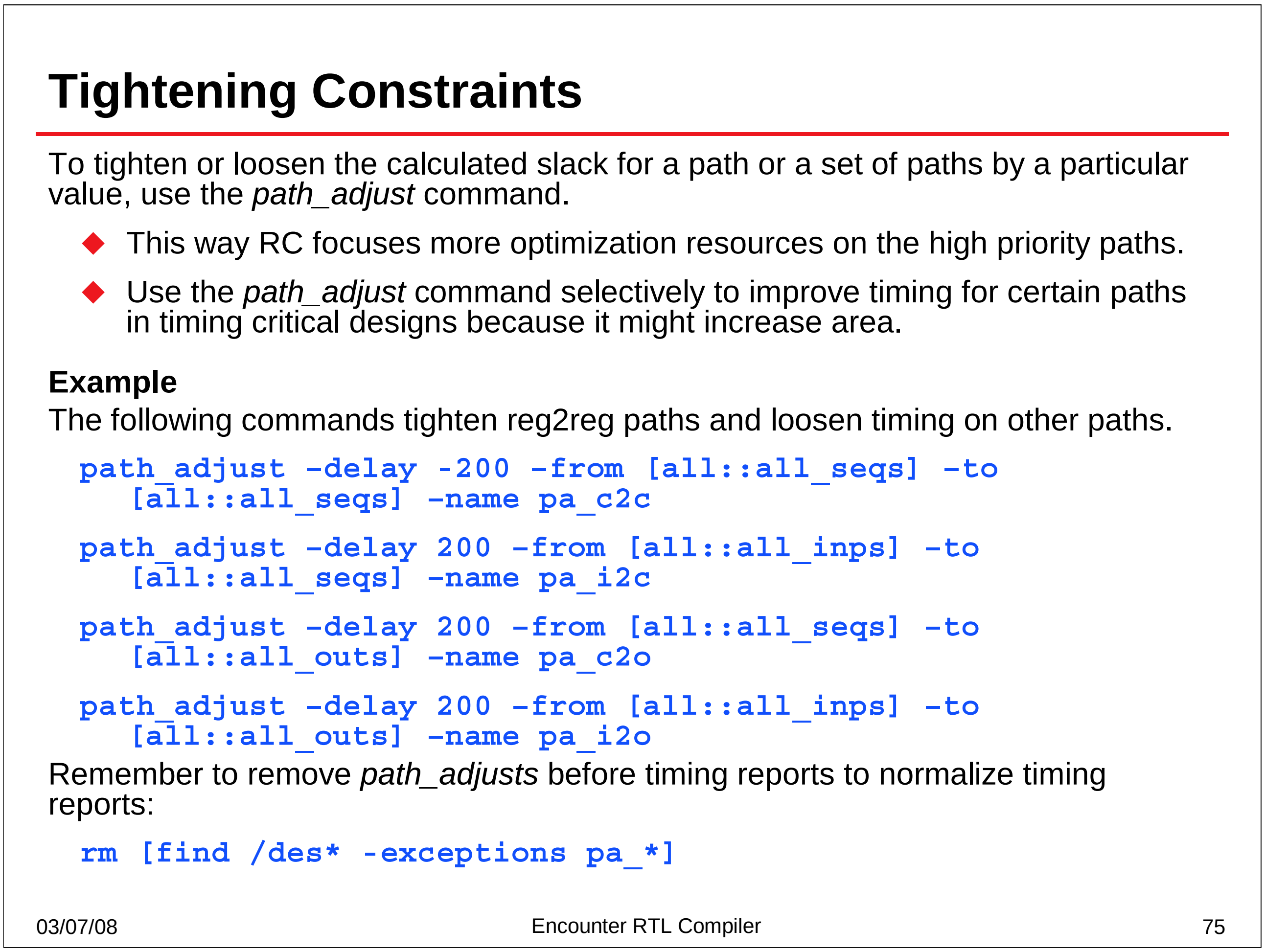
Example of Path Grouping

Example

```
define_cost_group -name sysclk  -design dtmf_chip
path_group -from [find / -clock CKG_VIT_CLK] \
      -group sysclk -n sysclk
```

Report

```
 Pin                                 Type      Fanout Load  Slew  Delay Arrival
                                                      (fF)  (ps)  (ps)    (ps)
--------------------------------------------------------------------------------
(clock CKG_VIT_CLK)             <<<  launch                                0 R
VIT_ACS0
…
ACS_DEC0[1]                     <<<  out port
(RC_vit_acsu_32.sdc_line_1551)     ext delay
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock CKG_VIT_CLK)                  capture                             800 R
--------------------------------------------------------------------------------
Cost Group    : 'sysclk' (path_group 'sysclk')
Start-point   : VIT_ACS0/cb_736/MY_REG_reg/clk
End-point     : ACS_DEC0[1]
```



Tightening Constraints

To tighten or loosen the calculated slack for a path or a set of paths by a particular value, use the *path_adjust* command.

- This way RC focuses more optimization resources on the high priority paths.
- Use the *path_adjust* command selectively to improve timing for certain paths in timing critical designs because it might increase area.

Example

The following commands tighten reg2reg paths and loosen timing on other paths.

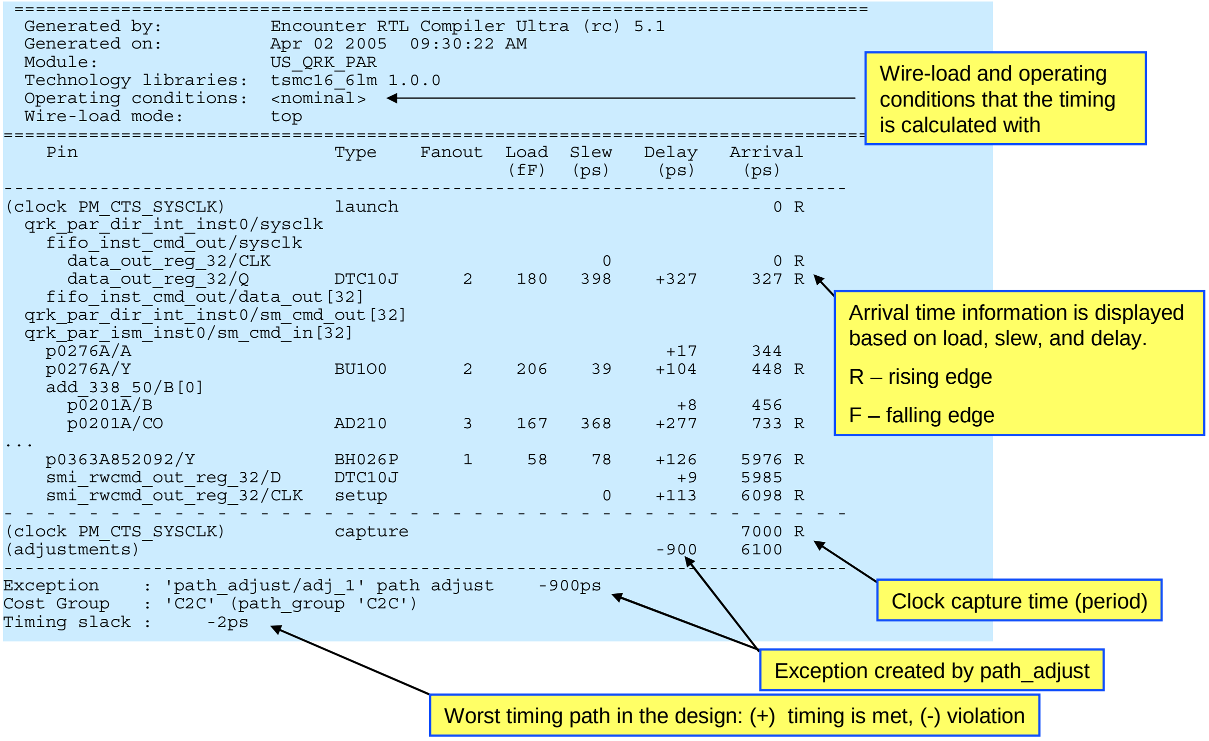
```
path_adjust -delay -200 -from [all::all_seqs] -to
   [all::all_seqs] -name pa_c2c

path_adjust -delay 200 -from [all::all_inps] -to
   [all::all_seqs] -name pa_i2c

path_adjust -delay 200 -from [all::all_seqs] -to
   [all::all_outs] -name pa_c2o

path_adjust -delay 200 -from [all::all_inps] -to
   [all::all_outs] -name pa_i2o
```

Remember to remove *path_adjusts* before timing reports to normalize timing reports:

```
rm [find /des* -exceptions pa_*]
```

You must include *load_etc.tcl* file for the *all::all_** commands to work.

Path Adjust: STA Using `report timing`



```
 ================================================================================
  Generated by:           Encounter RTL Compiler Ultra (rc) 5.1
  Generated on:           Apr 02 2005  09:30:22 AM
  Module:                 US_QRK_PAR
  Technology libraries:   tsmc16_6lm 1.0.0
  Operating conditions:   <nominal>
  Wire-load mode:         top
==================================================================================
    Pin                        Type     Fanout  Load  Slew   Delay   Arrival
                                               (fF)  (ps)    (ps)    (ps)
--------------------------------------------------------------------------------
(clock PM_CTS_SYSCLK)          launch                                    0 R
  qrk_par_dir_int_inst0/sysclk
    fifo_inst_cmd_out/sysclk
      data_out_reg_32/CLK                               0                0 R
      data_out_reg_32/Q        DTC10J      2    180   398    +327      327 R
    fifo_inst_cmd_out/data_out[32]
  qrk_par_dir_int_inst0/sm_cmd_out[32]
  qrk_par_ism_inst0/sm_cmd_in[32]
    p0276A/A                                                  +17      344
    p0276A/Y                   BU1O0       2    206    39    +104      448 R
    add_338_50/B[0]
      p0201A/B                                                 +8      456
      p0201A/CO                AD210       3    167   368    +277      733 R
...
    p0363A852092/Y             BH026P      1     58    78    +126     5976 R
    smi_rwcmd_out_reg_32/D     DTC10J                          +9     5985
    smi_rwcmd_out_reg_32/CLK   setup                    0    +113     6098 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock PM_CTS_SYSCLK)          capture                                7000 R
(adjustments)                                                -900     6100
--------------------------------------------------------------------------------
Exception    : 'path_adjust/adj_1' path adjust    -900ps
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :     -2ps
```

Wire-load and operating conditions that the timing is calculated with

Arrival time information is displayed based on load, slew, and delay.

R – rising edge

F – falling edge

Clock capture time (period)

Exception created by path_adjust

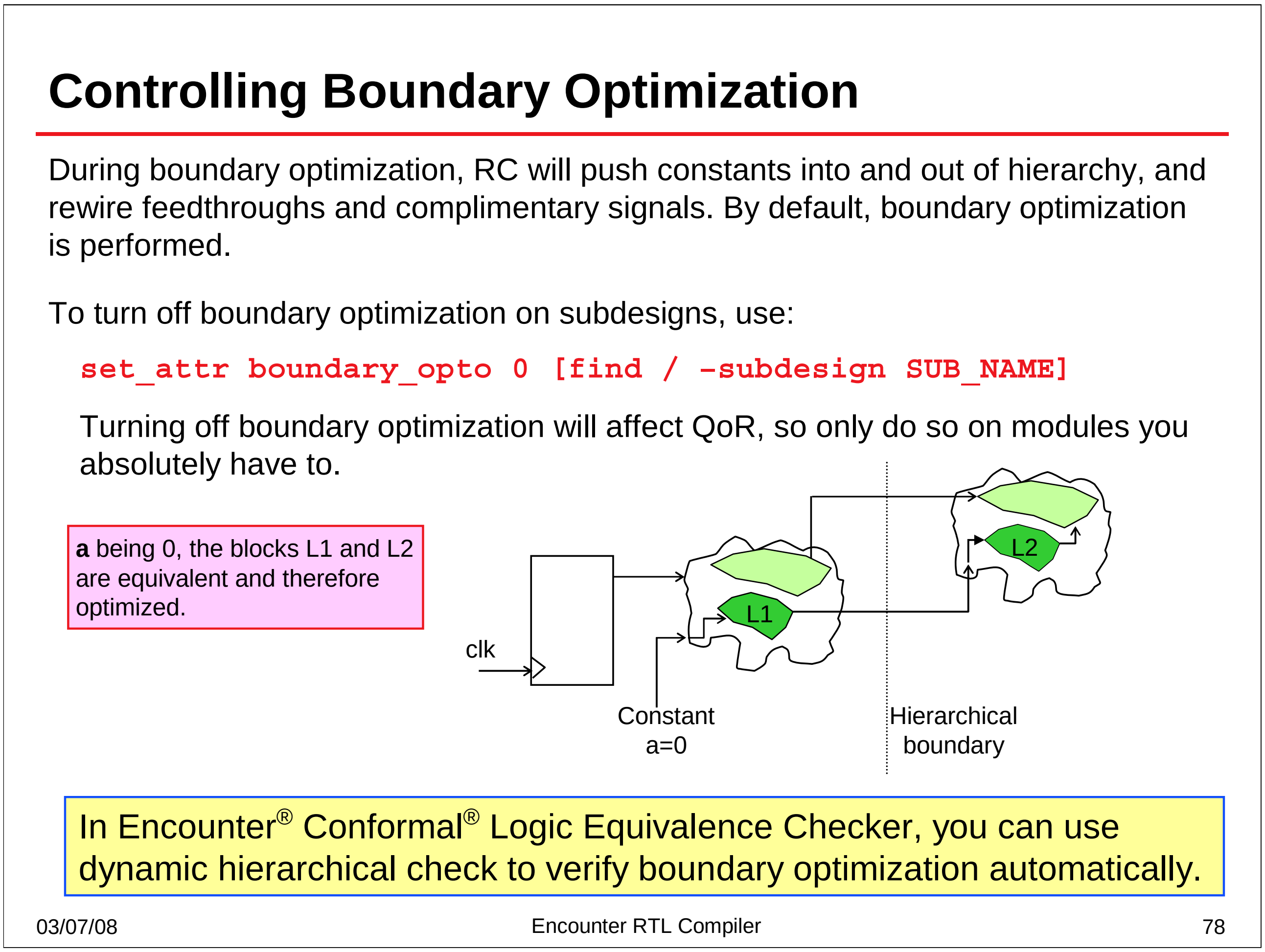
Worst timing path in the design: (+) timing is met, (-) violation

TNS/WNS Optimizations

To tell RC to optimize total negative slack (TNS), use:

```
set_attr tns_opto true /
```

- The default value of the *tns_opto* attribute is *false*.
- By default, RC optimizes the worst negative slack (WNS) in each cost group until it can no longer improve the timing of the worst paths in each cost group.
- Optimizing for WNS produces only better area but more timing violations. Therefore, if your design easily meets timing in the back end, then WNS optimization is the right approach.
- The TNS optimization approach produces fewer violations overall, meaning fewer issues in the place-and-route stage, assuming good timing correlation with the back end.
- As a general rule, always turn on TNS optimization, unless you have a reason not to.



Controlling Boundary Optimization

During boundary optimization, RC will push constants into and out of hierarchy, and rewire feedthroughs and complimentary signals. By default, boundary optimization is performed.

To turn off boundary optimization on subdesigns, use:

```
set_attr boundary_opto 0 [find / -subdesign SUB_NAME]
```

Turning off boundary optimization will affect QoR, so only do so on modules you absolutely have to.

a being 0, the blocks L1 and L2 are equivalent and therefore optimized.

In Encounter® Conformal® Logic Equivalence Checker, you can use dynamic hierarchical check to verify boundary optimization automatically.

Turning off boundary optimization will affect quality of results (QoR), so only do so on modules you have to. For example, any cell connected to an output port of a *boundary_opto false* subdesign, will be treated as connected to a primary output and will not be deleted.

Flops with data pins that are driven by constant values determined either through RTL assignment or via boundary optimization will be removed during optimization. To disable constant propagation, use

```
set_attr optimize_constant_0_flops false  (default is true)
set_attr optimize_constant_1_flops false  (default is false)
```

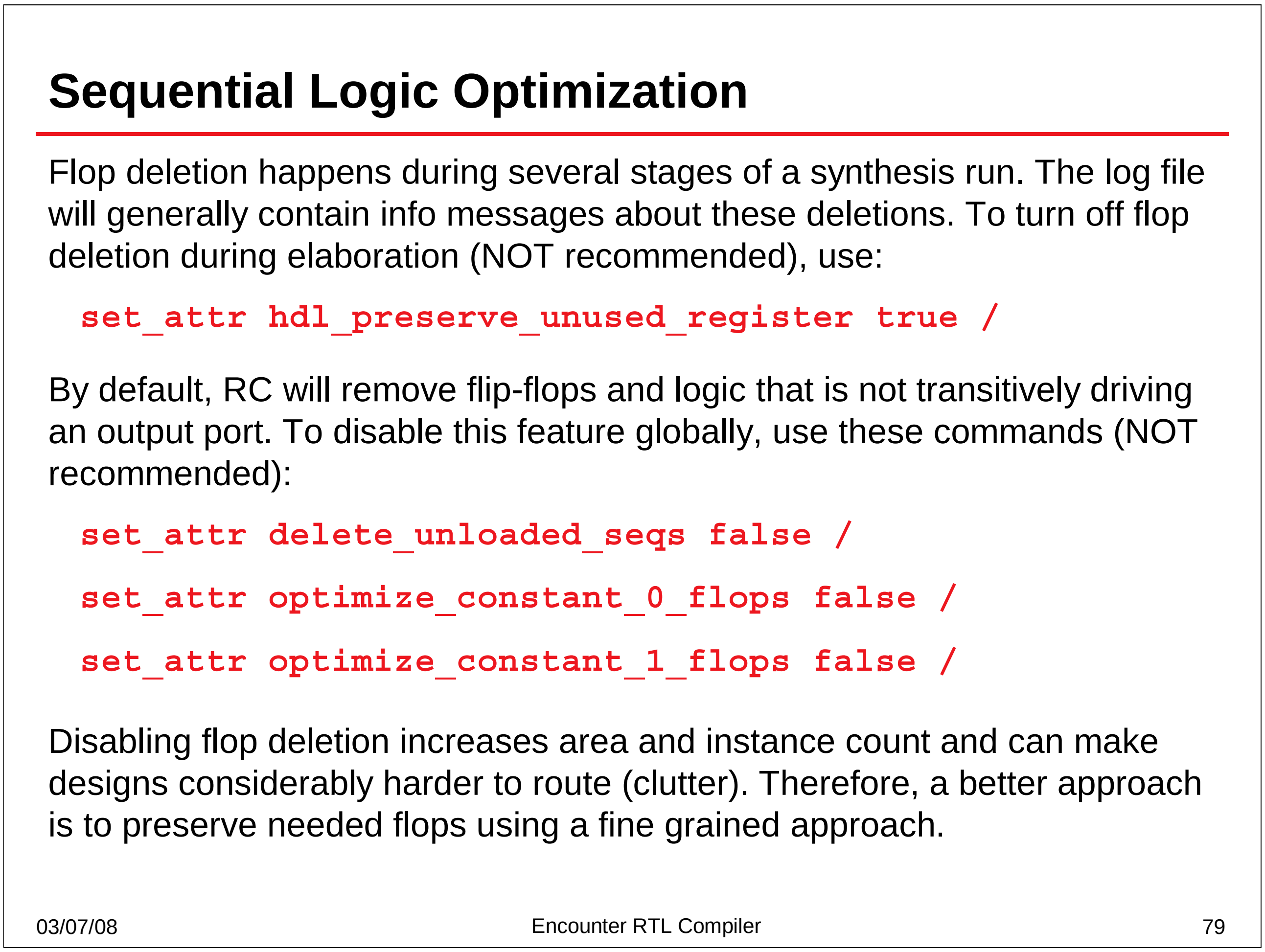
Rewiring of Equivalent Signals Across Hierarchy

If two outputs of a module are identical, the compiler might disconnect one of them and use the other output to drive the fanout logic for both.

The sequential elements are automatically inferred. You can take advantage of the complex flip-flops in the library with the sequential mapping feature to improve the cell count, area, or timing of your design.

To keep the synchronous feedback logic immediately in front of the sequential elements, enter the following:

```
set_attr hdl_ff_keep_feedback 1 [find / -hdl_arch DFF*]
```



Sequential Logic Optimization

Flop deletion happens during several stages of a synthesis run. The log file will generally contain info messages about these deletions. To turn off flop deletion during elaboration (NOT recommended), use:

```
set_attr hdl_preserve_unused_register true /
```

By default, RC will remove flip-flops and logic that is not transitively driving an output port. To disable this feature globally, use these commands (NOT recommended):

```
set_attr delete_unloaded_seqs false /

set_attr optimize_constant_0_flops false /

set_attr optimize_constant_1_flops false /
```

Disabling flop deletion increases area and instance count and can make designs considerably harder to route (clutter). Therefore, a better approach is to preserve needed flops using a fine grained approach.

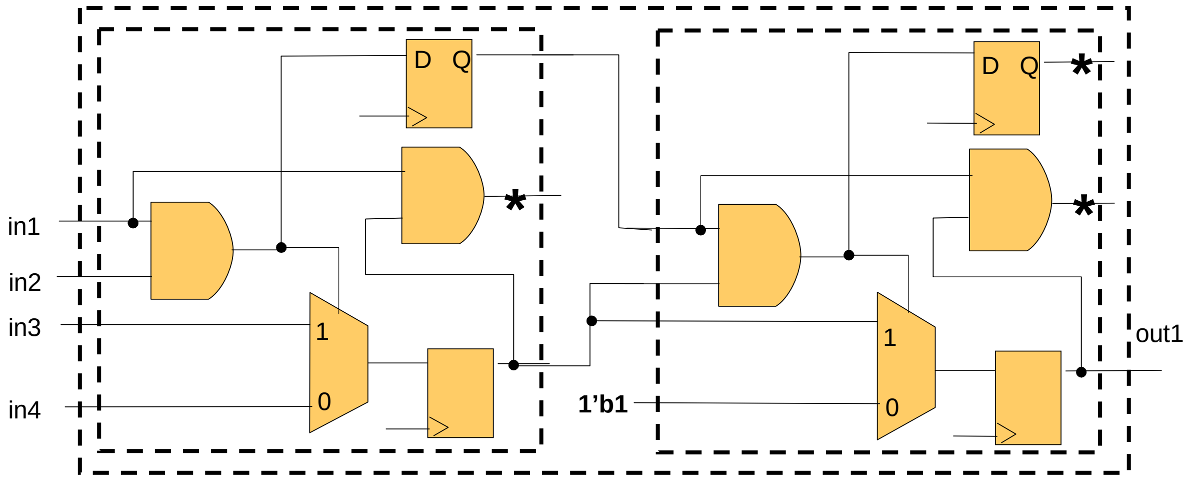
Disabling flop deletion increases area and instance count. A better (more granular) approach is to use *preserve* on instances that need to be preserved. Turning off these optimizations results in higher congestion and in difficulty closing timing downstream.

Pruning Logic Driving Unused Pins

By default, logic that drives unused (unloaded) hierarchical pins will be optimized away.

To preserve this logic, use this command:

```
set_attr prune_unused_logic false <path to pins>
```

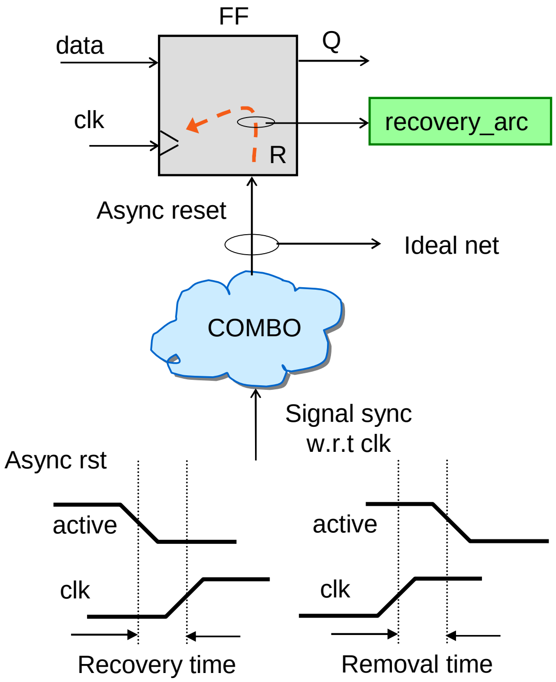


Optimizing Logic Ending at Asynchronous Reset

To optimize the combinational logic on the reset pin for timing, use the *recovery_arc* in the library of the flop as a setup arc.

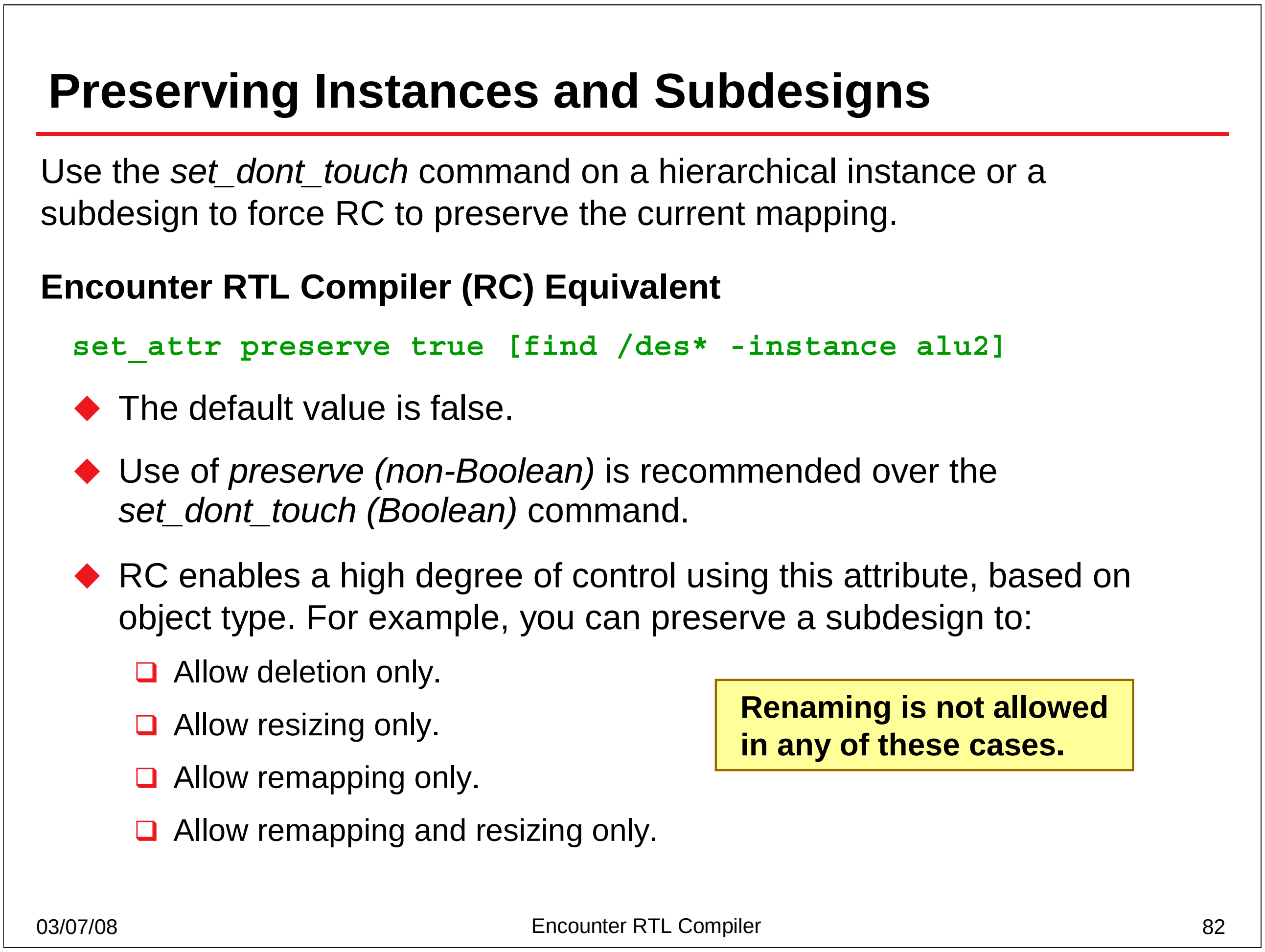
```
set_attr time_recovery_arcs true /
```

This root attribute is by default set to ***false***. By default, this path will be treated as asynchronous and not optimized.



Example

	attr **true**	attr **false**
Gates	190	138
Area	5279	3835
Path time	3000 (sync) meets timing	4561 (async: unconstrained)



Preserving Instances and Subdesigns

Use the *set_dont_touch* command on a hierarchical instance or a subdesign to force RC to preserve the current mapping.

Encounter RTL Compiler (RC) Equivalent

```
set_attr preserve true [find /des* -instance alu2]
```

- The default value is false.
- Use of *preserve (non-Boolean)* is recommended over the *set_dont_touch (Boolean)* command.
- RC enables a high degree of control using this attribute, based on object type. For example, you can preserve a subdesign to:
  - Allow deletion only.
  - Allow resizing only.
  - Allow remapping only.
  - Allow remapping and resizing only.

Renaming is not allowed in any of these cases.

Options for Preserving Instances in RC

```
set_attr preserve map_size_ok [find / -instance g1]
```

- Allows resizing, mapping, and remapping of a sequential instance during optimization, but not renaming or deleting it.

```
set_attr preserve size_ok [find / -instance g1]
```

- Enables the compiler to preserve the instance g1, but also to give the flexibility to size it.

```
set_attr preserve delete_ok [find / -instance g1]
```

- g1 can be deleted, but it will not be renamed or remapped.

```
set_attr preserve size_delete_ok [find / -instance g1]
```

- g1 can be resized or deleted, but it will not be renamed or remapped.

```
set_attr delete_unloaded_insts  false /
```

- Set this root attribute to *false* to prevent deletion of unloaded instances.

Grouping and Ungrouping of Hierarchy

Grouping

Create hierarchy to partition your design with the *group* command.

Example

```
group -group_name CRITICAL_GROUP [find /
-instance I1] [find / -instance I2]
```

Ungrouping

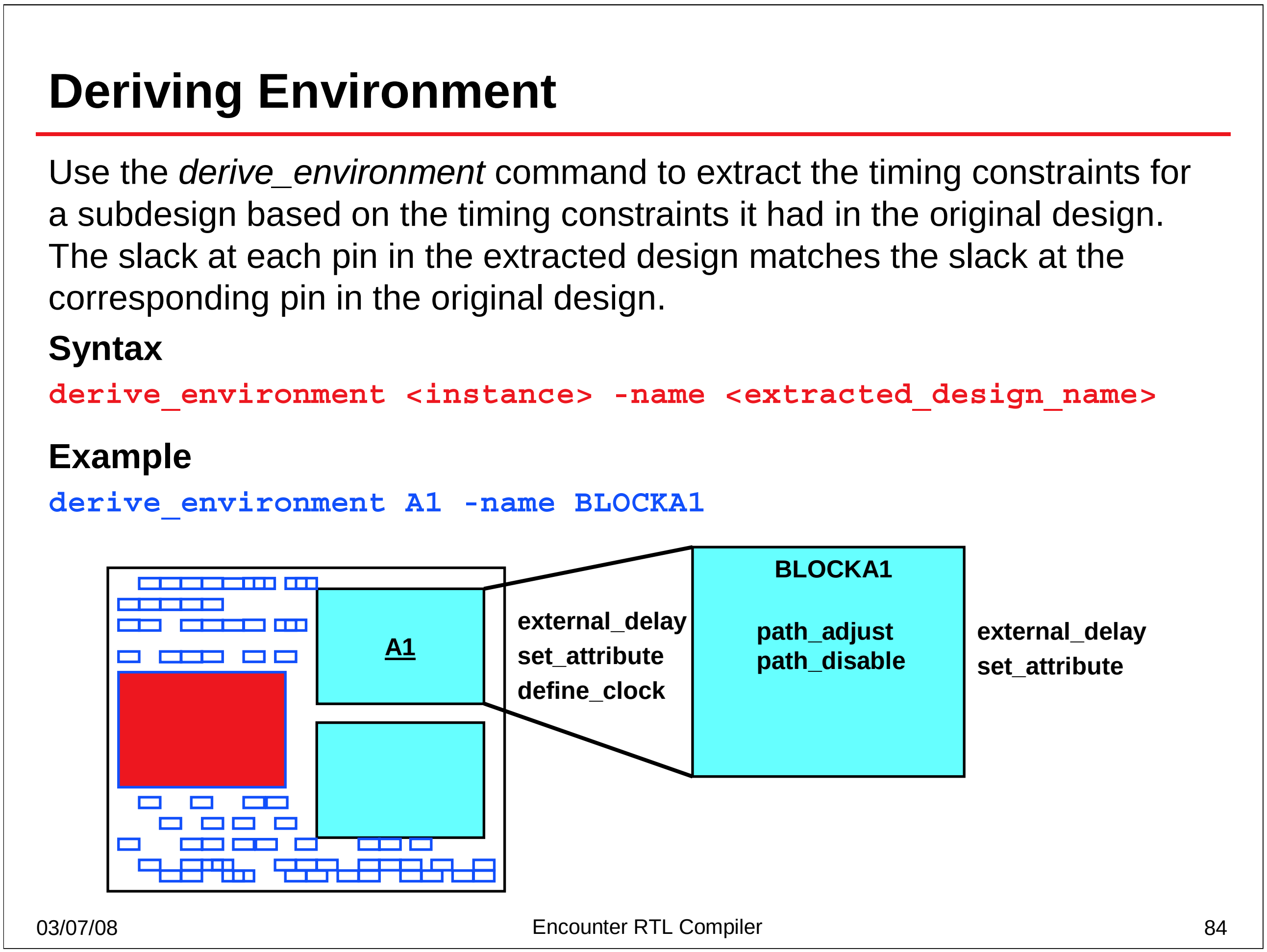
Manually dissolve an instance of the hierarchy with the *ungroup* command. You can also recursively ungroup all hierarchical instances of a specified size.

Examples

```
ungroup [find / -instance CRITICAL_GROUP]
```

```
ungroup -threshold 500
```

Hierarchical ungrouping can improve timing by giving RC a wider scope of optimization.



Deriving Environment

Use the *derive_environment* command to extract the timing constraints for a subdesign based on the timing constraints it had in the original design. The slack at each pin in the extracted design matches the slack at the corresponding pin in the original design.

Syntax

```
derive_environment <instance> -name <extracted_design_name>
```

Example

```
derive_environment A1 -name BLOCKA1
```

A1

external_delay
set_attribute
define_clock

BLOCKA1

path_adjust
path_disable

external_delay
set_attribute

The steps below illustrate how to synthesize a submodule:

- elaborate module_top
- synthesize -to_mapped -effort low
- derive_environment -name <new_top> -instance <new_top_instance_name>

```
# At this stage you have two new designs at the top level:
module_top and new_top.
```

- write_script new_top > new_top.g

```
# For best results, remove the new_top module, re-read and
synthesize the design.
```

- rm new_top
- read_hdl <read_RTL_files>
- elaborate <new_top>
- include new_top.g
- synthesize

Bottom-Up Design Flow

- Load and constrain the top level.

```
read_hdl $FILE_LIST
elaborate
```

- Map the top level.

```
synthesize -to_mapped -effort low
```

- Derive constraints for the block.

```
derive_environment DTMF_INST/DMA_INST
```

- Synthesize the block with a high effort.

```
cd dma
synthesize -to_mapped -effort high dma
```

- Go to the top level and link the new block.

```
cd /designs/dtmf_chip
report timing
change_link -inst [find / -inst DMA_INST]
-design /designs/dma
```

- Reapply any multicycle constraints, because they might be removed during *change_link*.
- Preserve instances as needed.

```
set_attr preserve [find / -inst DMA_INST]
```

- Resynthesize the top level with the proper effort level.

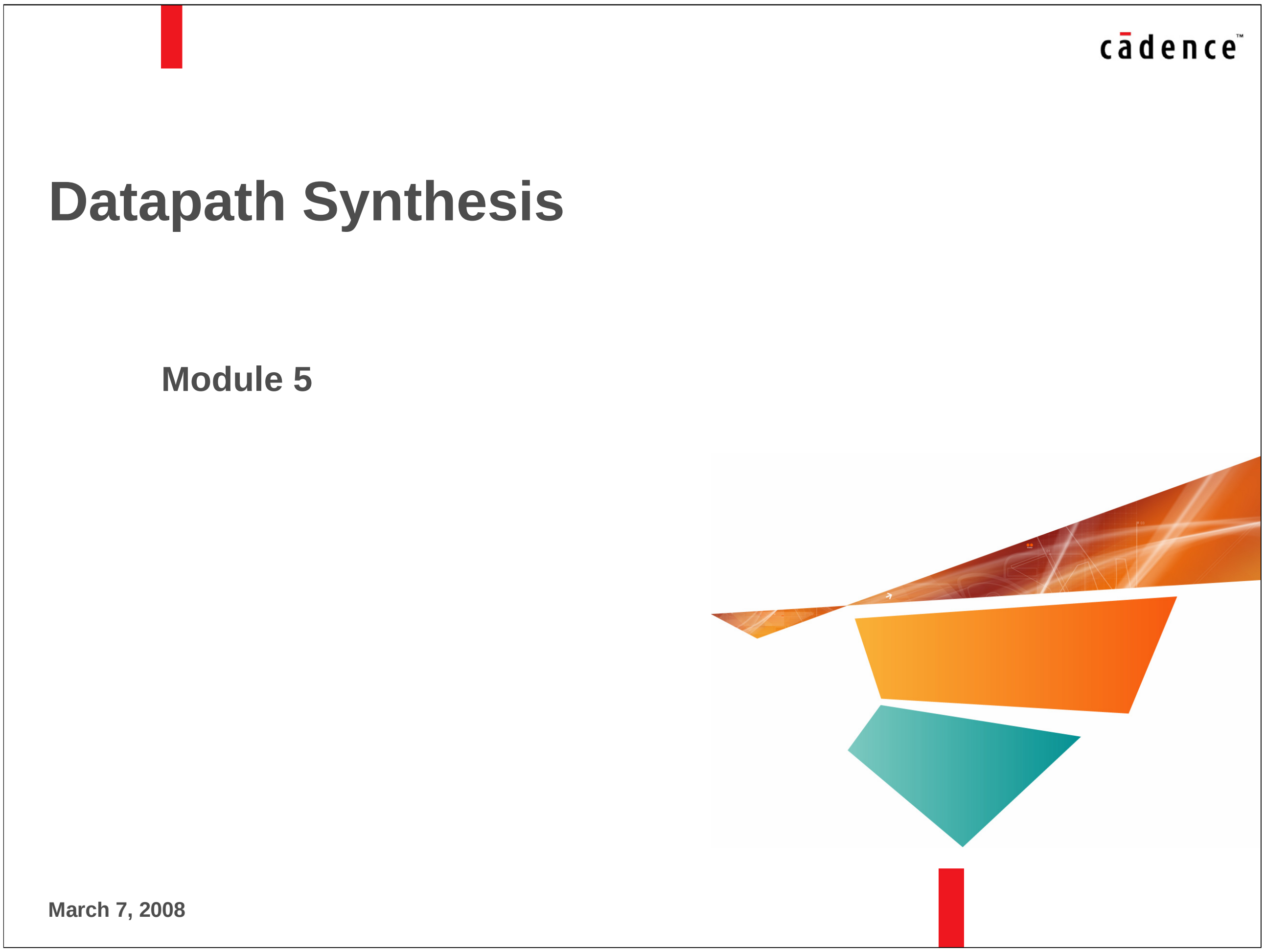
```
synthesize -to_mapped -effort medium
```

- Report, analyze, and save.

Lab Exercises

Lab 4-1 Exploring Optimization Strategies

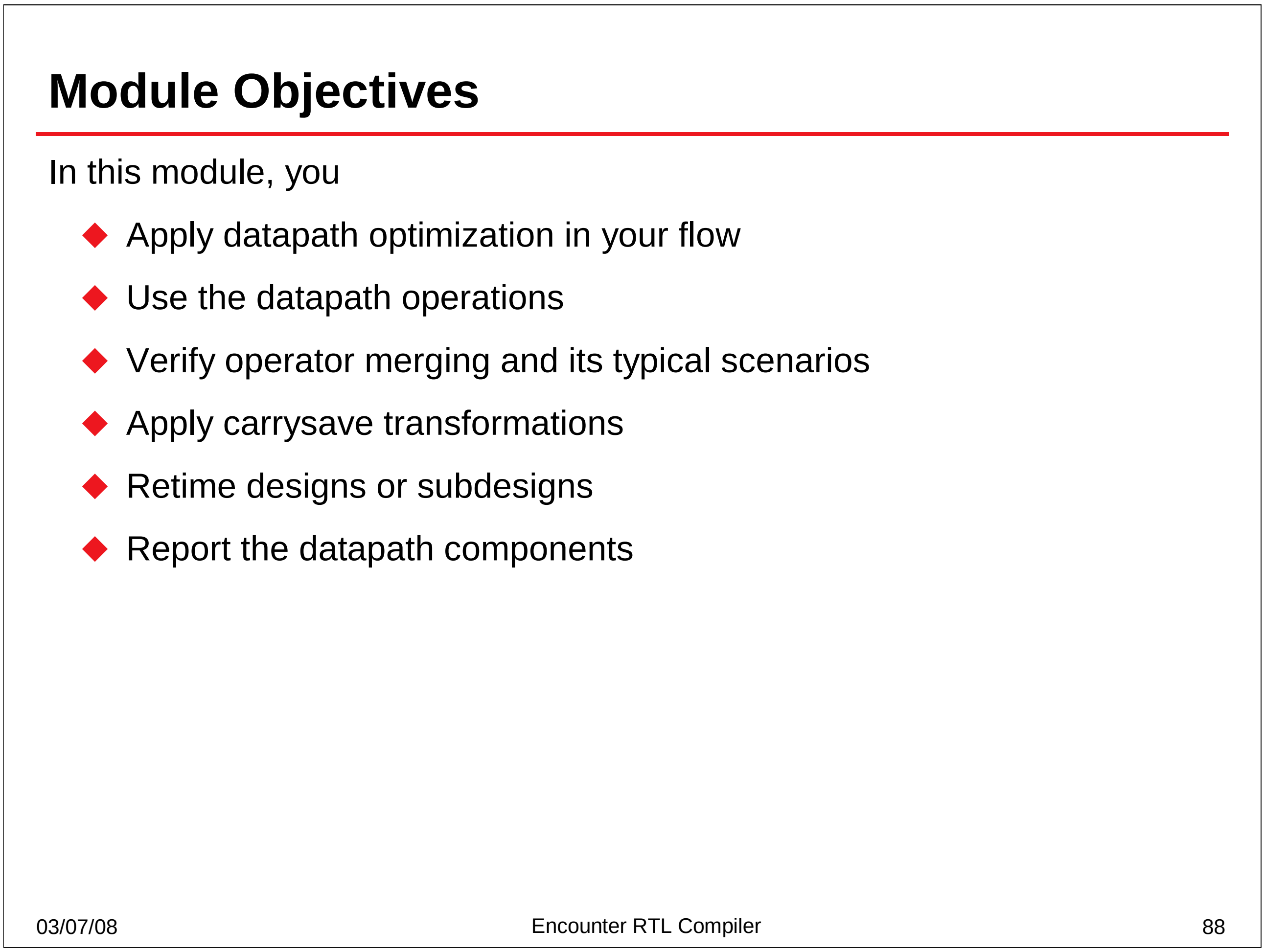
- Setting Up the Environment
- Setting Optimization Attributes
- Defining Cost Groups
- Running Generic Synthesis
- Mapping to a Target Library
- Running Incremental Synthesis
- Generating and Analyzing Reports
- Saving Synthesis Results



Datapath Synthesis

Module 5

March 7, 2008

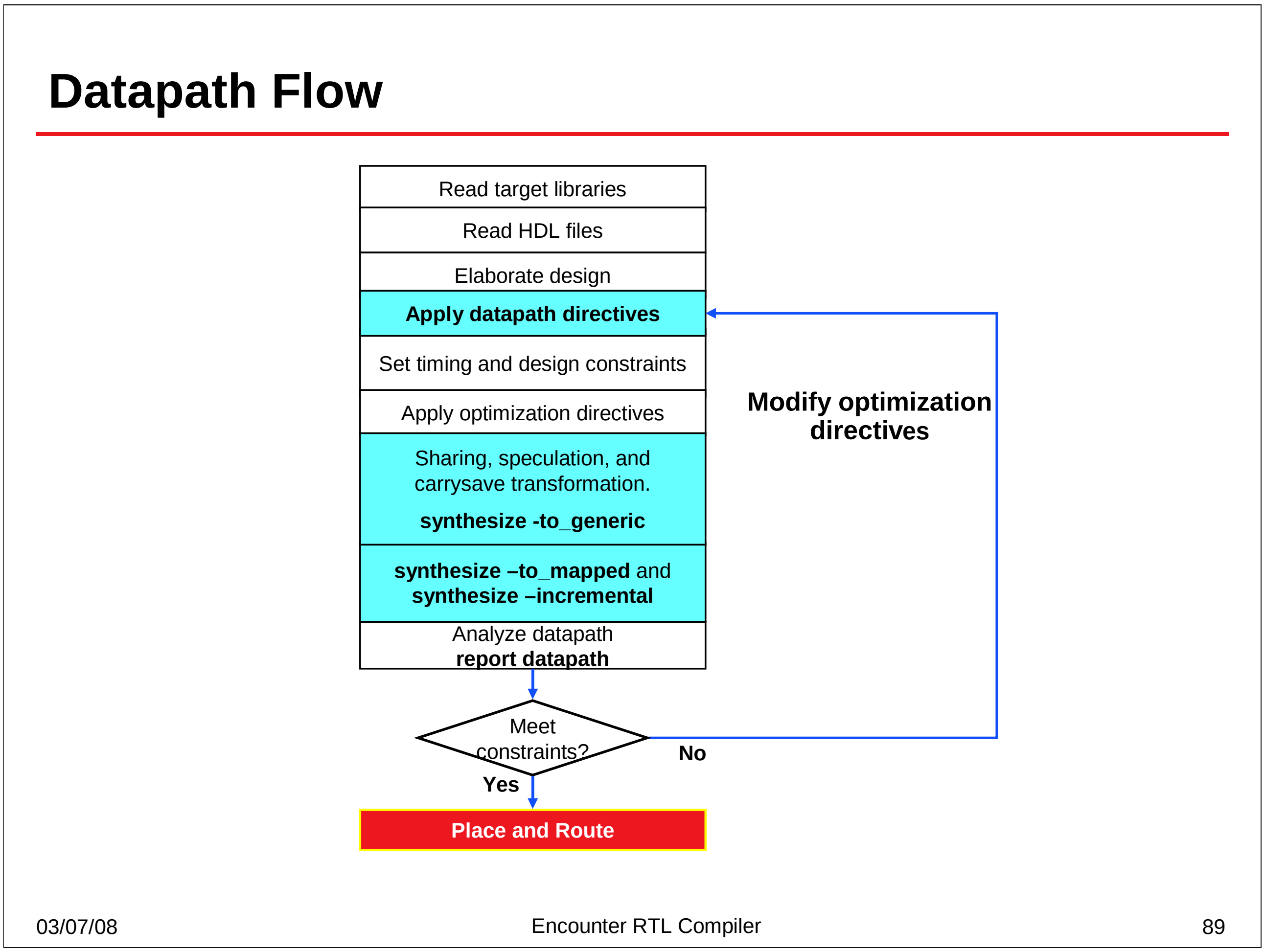


Module Objectives

In this module, you

- Apply datapath optimization in your flow
- Use the datapath operations
- Verify operator merging and its typical scenarios
- Apply carrysave transformations
- Retime designs or subdesigns
- Report the datapath components

03/07/08 Encounter RTL Compiler 88



Datapath Flow

Read target libraries

Read HDL files

Elaborate design

Apply datapath directives

Set timing and design constraints

Apply optimization directives

Sharing, speculation, and carrysave transformation.

synthesize -to_generic

synthesize –to_mapped and **synthesize –incremental**

Analyze datapath
report datapath

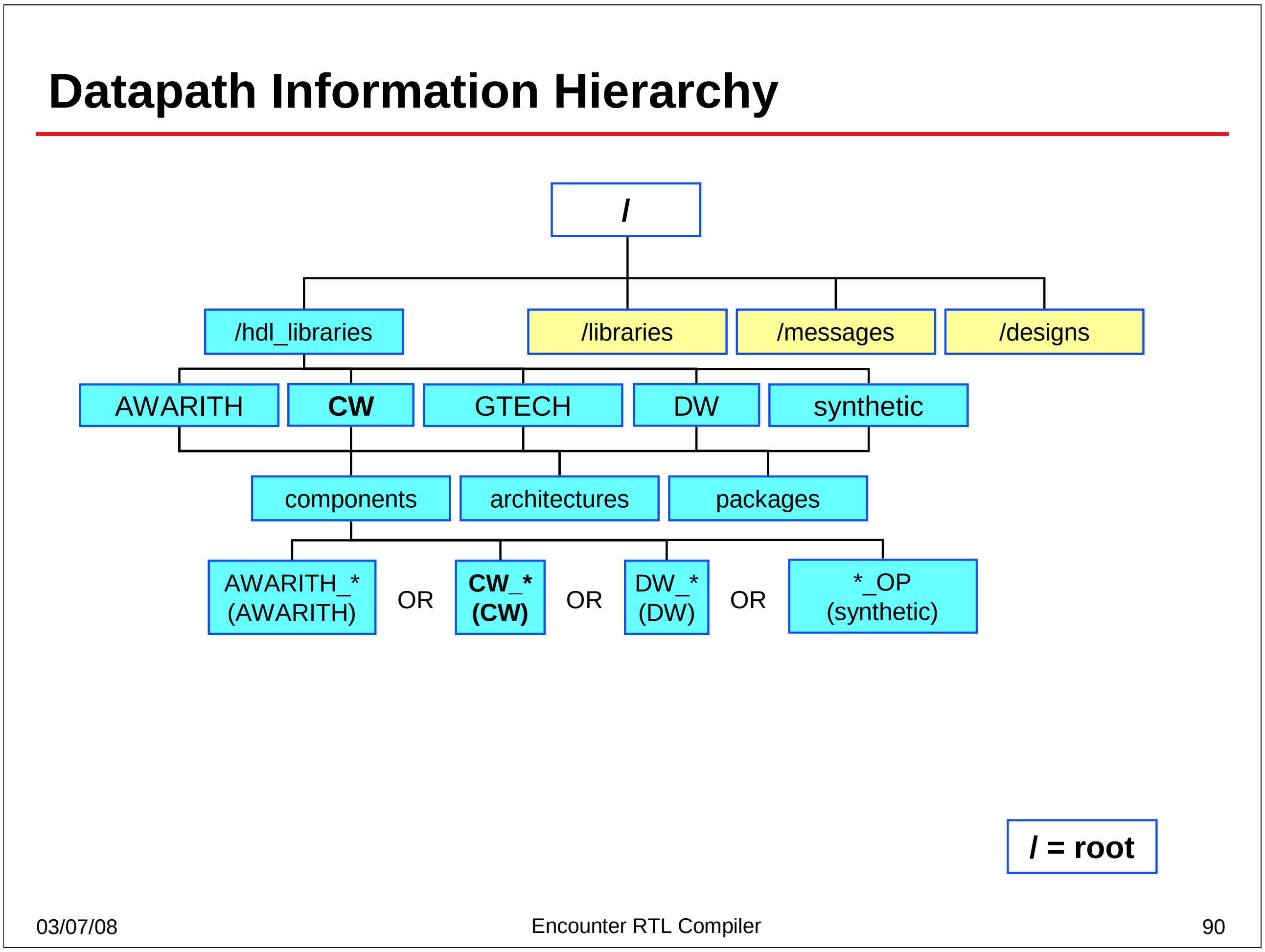
Meet constraints?

No

Modify optimization directives

Yes

Place and Route



Datapath Information Hierarchy

/

/hdl_libraries

/libraries

/messages

/designs

AWARITH

CW

GTECH

DW

synthetic

components

architectures

packages

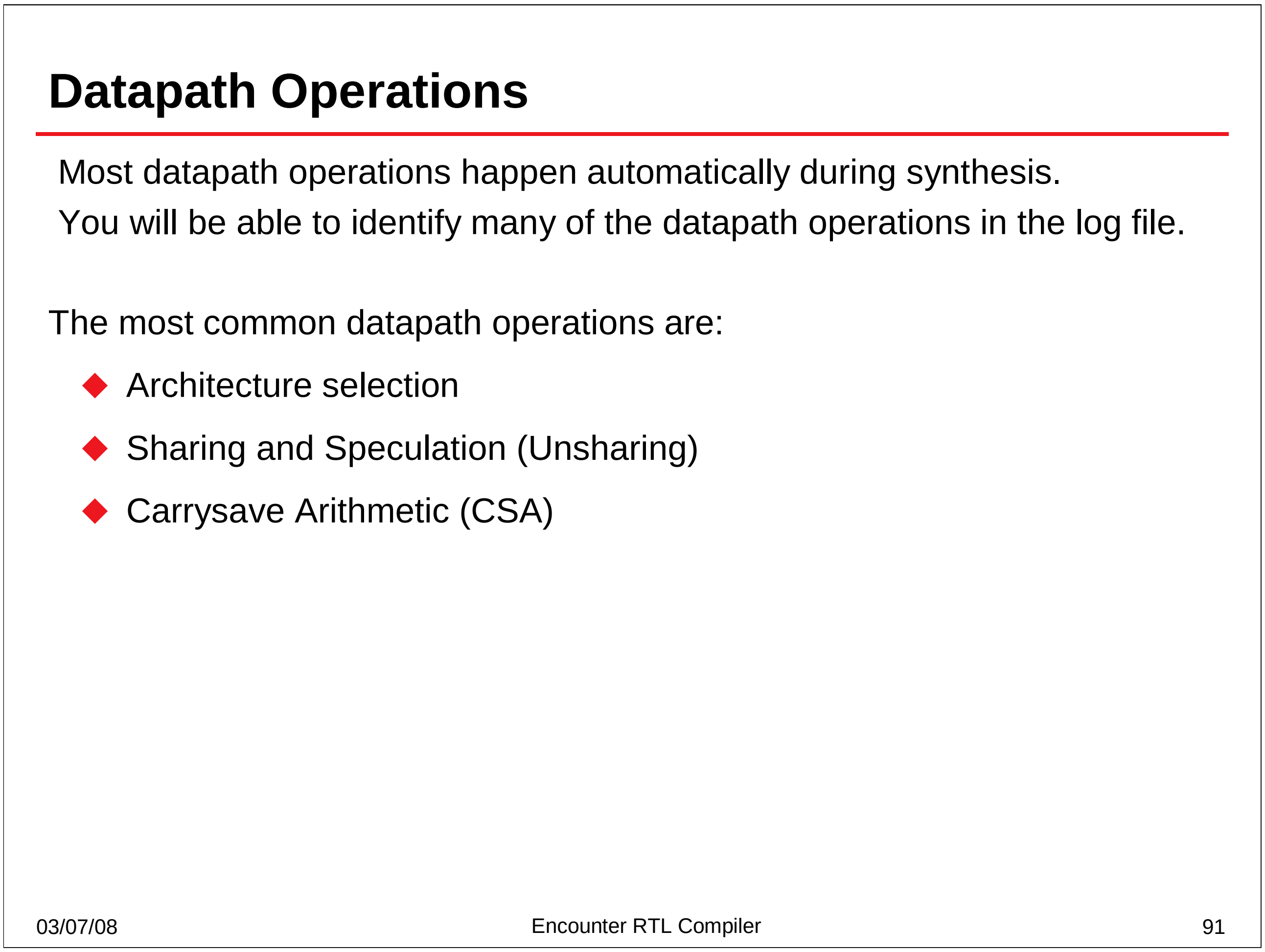
AWARITH_*
(AWARITH) OR **CW_*
(CW)** OR DW_*
(DW) OR *_OP
(synthetic)

/ = root

03/07/08 Encounter RTL Compiler 90

The following functions are part of the datapath package:

- ChipWare Components, Designware, Ambitware, and GTECH cells
- Extended VHDL and Verilog language interfaces for concise coding of complex datapath designs
- Primitive functions such as $abs(int), $blend(X0,X1,alpha,alpha1), $carrysave(int), $intround(int,posint), $lead0(bin), $lead1(bin), $rotatel(in, rotate), $rotater(in, rotate), and $round(in, pos)
- Parameter Functions such as $log2(int), $max(int, int) and $min(int, int)



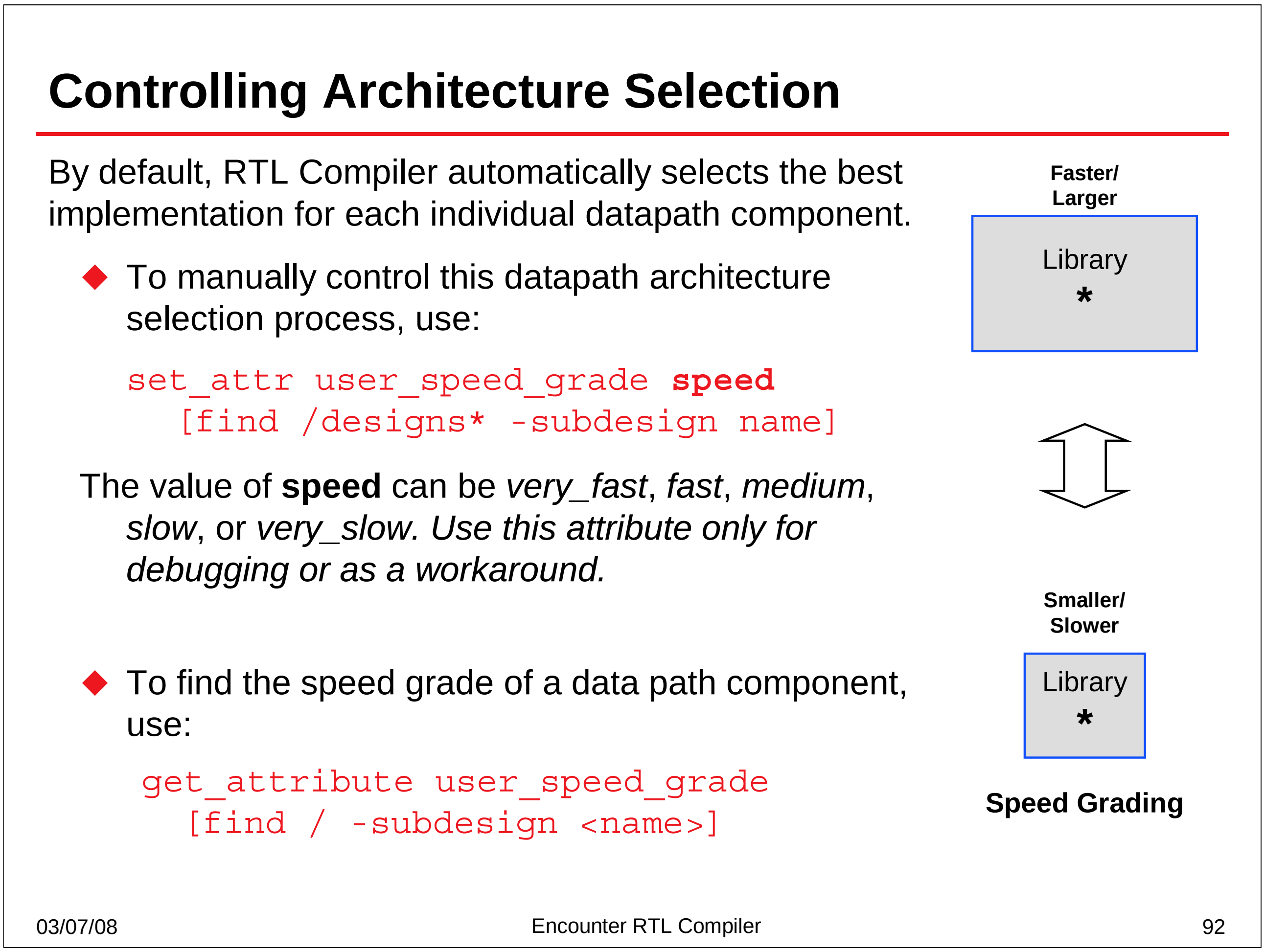
Datapath Operations

Most datapath operations happen automatically during synthesis.

You will be able to identify many of the datapath operations in the log file.

The most common datapath operations are:

- Architecture selection
- Sharing and Speculation (Unsharing)
- Carrysave Arithmetic (CSA)



Controlling Architecture Selection

By default, RTL Compiler automatically selects the best implementation for each individual datapath component.

- To manually control this datapath architecture selection process, use:

```
set_attr user_speed_grade speed
   [find /designs* -subdesign name]
```

The value of **speed** can be *very_fast*, *fast*, *medium*, *slow*, or *very_slow. Use this attribute only for debugging or as a workaround.*

- To find the speed grade of a data path component, use:

```
get_attribute user_speed_grade
   [find / -subdesign <name>]
```

Faster/
Larger

Library
*

Smaller/
Slower

Library
*

Speed Grading

You can only set the *user_speed_grade* attribute on the datapath subdesign created by RTL Compiler during elaboration and not the subdesign from your RTL.

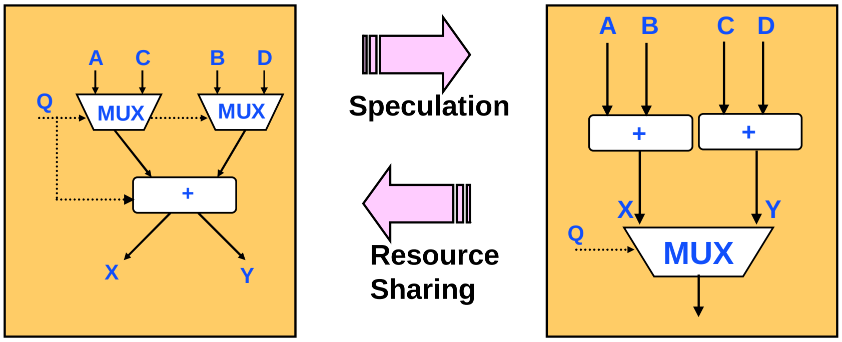
Sharing and Speculation

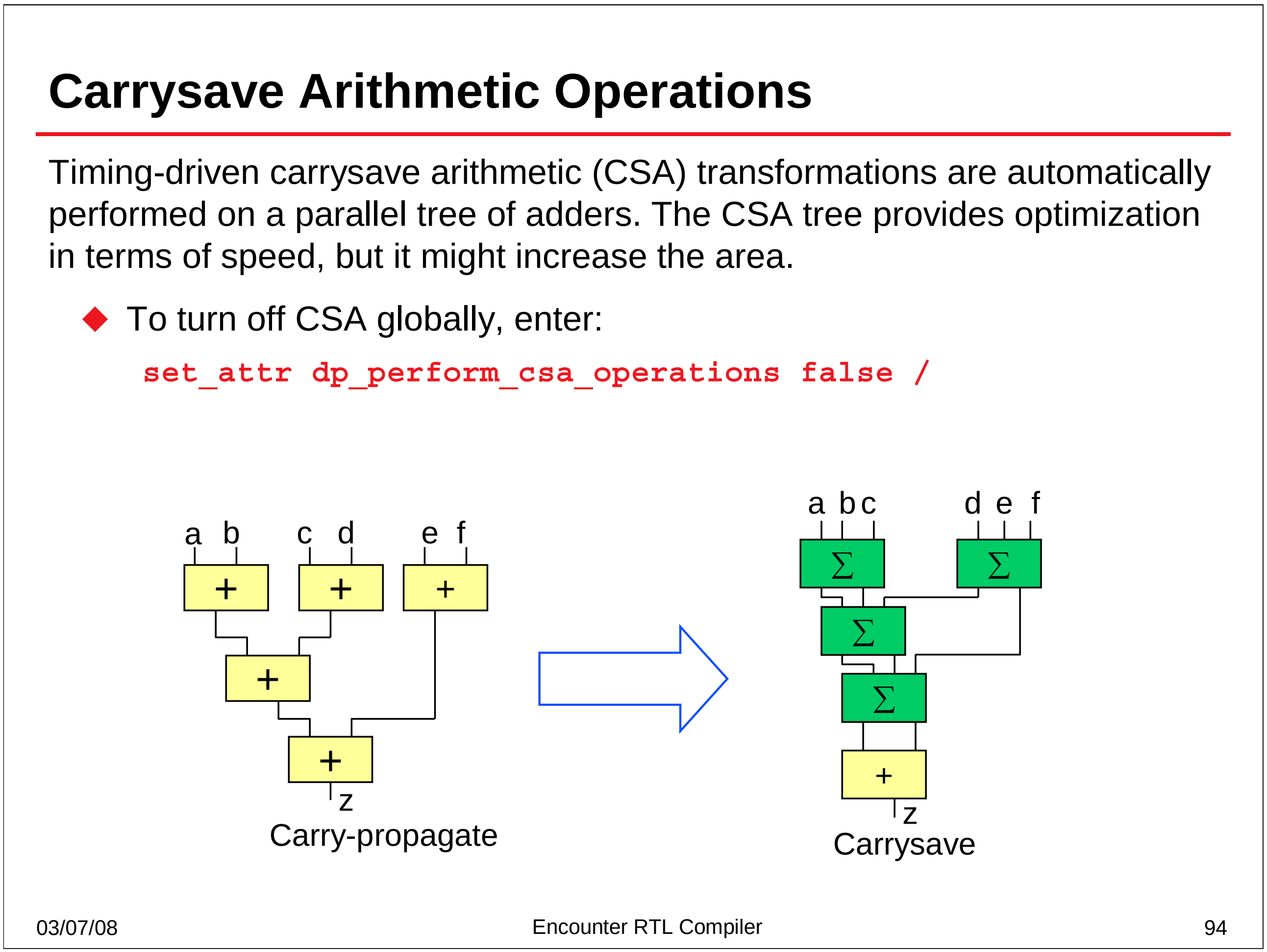
Sharing and speculation take place automatically during synthesis when you use the *synthesize -to_generic -effort high* command (timing driven mode). The compiler automatically shares and unshares (speculates) the datapath instances based on the optimization criteria.

- To globally control sharing and speculation when applying other effort levels of generic optimization, use these commands:

```
set_attr dp_perform_sharing_operations true|false /
set_attr dp_perform_speculation_operations true|false /
```

```
if (Q ='0')
 x = a + b;
else
 y = c + d;
```





Carrysave Arithmetic Operations

Timing-driven carrysave arithmetic (CSA) transformations are automatically performed on a parallel tree of adders. The CSA tree provides optimization in terms of speed, but it might increase the area.

- To turn off CSA globally, enter:

```
set_attr dp_perform_csa_operations false /
```

Carry-propagate

Carrysave

Datapath operators are merged in scenarios such as the following :

- Any combination of Vector Sum, Sum-of-Product, and Product-of-Sum, including intermediary inverted signals.

```
assign cs = a + b + c + d; assign y = cs * p + q;
```

- Comparator

```
assign p = a + b; assign q = c + d; assign is_greater = (p > q);
```

- Multiple Fanout

```
cs = a * b; x = cs + c; y = cs + d;
```

- Multiplexers

```
assign tmp = s ? a*b + c*d : a*c + b*d; assign z = tmp + e;
```

- Inverter

```
wire [16:0] tmp = a + b; assign z = ~tmp + c;
```

- Truncated CSA

```
assign p = a + b; assign q = c + d; assign r = p + q;
assign y = r[17:8];
```

Fine-tuning Datapath Operations

To control sharing on subdesigns, enter:

```
set_attr allow_sharing_subdesign {true|false}
[find / -subdesign name]
```

To control speculation on subdesigns, enter:

```
set_attr allow_speculation_subdesign {true|false}
[find / -subdesign name]
```

To control CSA on subdesigns, enter:

```
set_attr allow_csa_subdesign {true|false}
[find / -subdesign name]
```

Optimizing Datapath for Area

To improve area during datapath optimization, use the *dp_area_mode* attribute.

```
set_attribute dp_area_mode true /
```

Setting this attribute to true achieves area gain by performing the following:

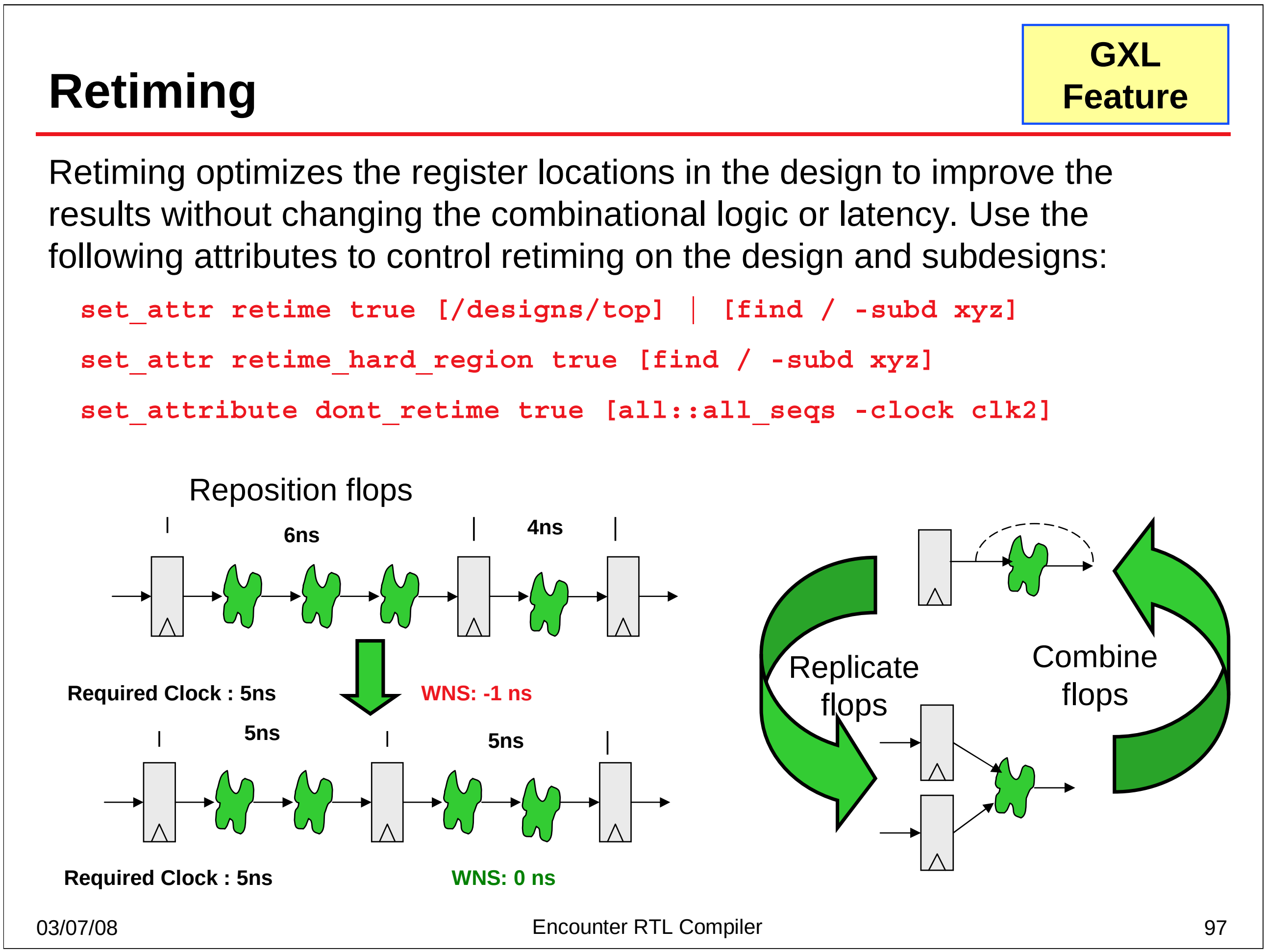
- Full adder cell mapping
- Conservative *carry-save adder* (CSA) transformations
- Sharing before CSA transformations

Perform architecture downsizing after mapping by setting the *dp_postmap_downsize* attribute to true.

```
set_attribute dp_postmap_downsize true /
```

This attribute is effective only during incremental optimization. However, there is a potential increase in run time.

By default, resource sharing is performed after CSA transformation. If you set the *dp_area_mode* attribute to true, sharing is performed before the CSA tree operation.



Retiming

GXL Feature

Retiming optimizes the register locations in the design to improve the results without changing the combinational logic or latency. Use the following attributes to control retiming on the design and subdesigns:

```
set_attr retime true [/designs/top] | [find / -subd xyz]
set_attr retime_hard_region true [find / -subd xyz]
set_attribute dont_retime true [all::all_seqs -clock clk2]
```

03/07/08 Encounter RTL Compiler 97

The Encounter® RTL Compiler retiming combines generic retiming with the optimization of combinational logic.

The following attributes help control the retiming of your design:

```
set_attr retime_async_reset true  /
```

Retime flops with asynchronous reset or set (by default they get *dont_retime*), increases runtime.

```
set_attr retime_optimize_reset true  /
```

Replace asynchronous reset or replace asynchronous set flops whose reset or set values evaluate to *dont_care* with simple flops.

```
set_attr retime_hard_region true [subdesign]
```

Retime regions set as hard region using the *hard_region* attribute..

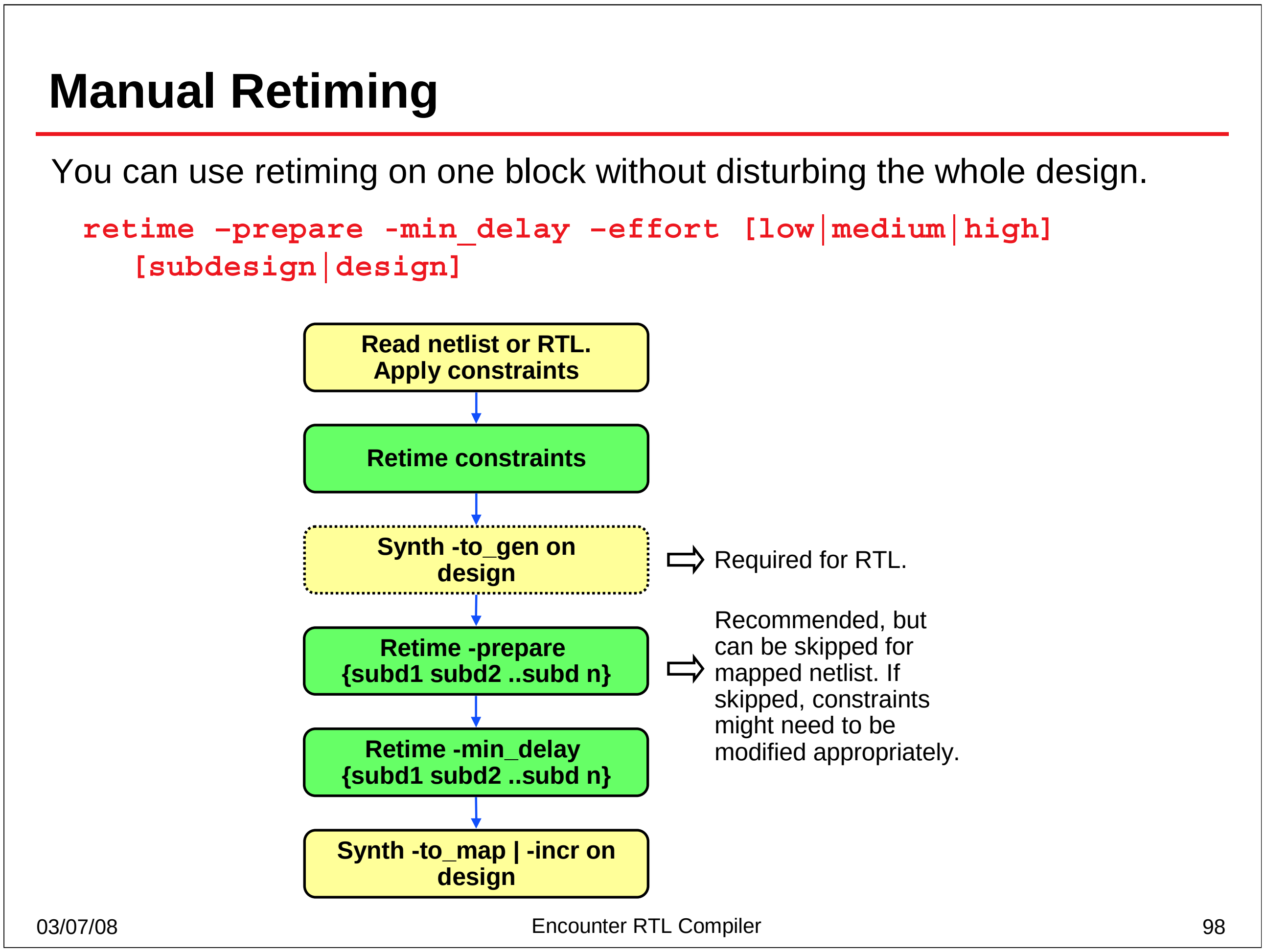
```
set_attr retime_reg_naming_suffix "flop"  /
```

Results in retime flops as *retime_1_flop* instead of the default, *retime _1_reg*.

```
set_attr trace_retime true <flops>
```

Select flops with *trace_retime true* for tracing the original name of the flops. Use on limited flops only. To get the status of these flops, use:

```
get_attr retime_original_registers
```



Manual Retiming

You can use retiming on one block without disturbing the whole design.

```
retime -prepare -min_delay -effort [low|medium|high]
   [subdesign|design]
```

Read netlist or RTL.
Apply constraints

Retime constraints

Synth -to_gen on design

Retime -prepare {subd1 subd2 ..subd n}

Retime -min_delay {subd1 subd2 ..subd n}

Synth -to_map | -incr on design

⇨ Required for RTL.

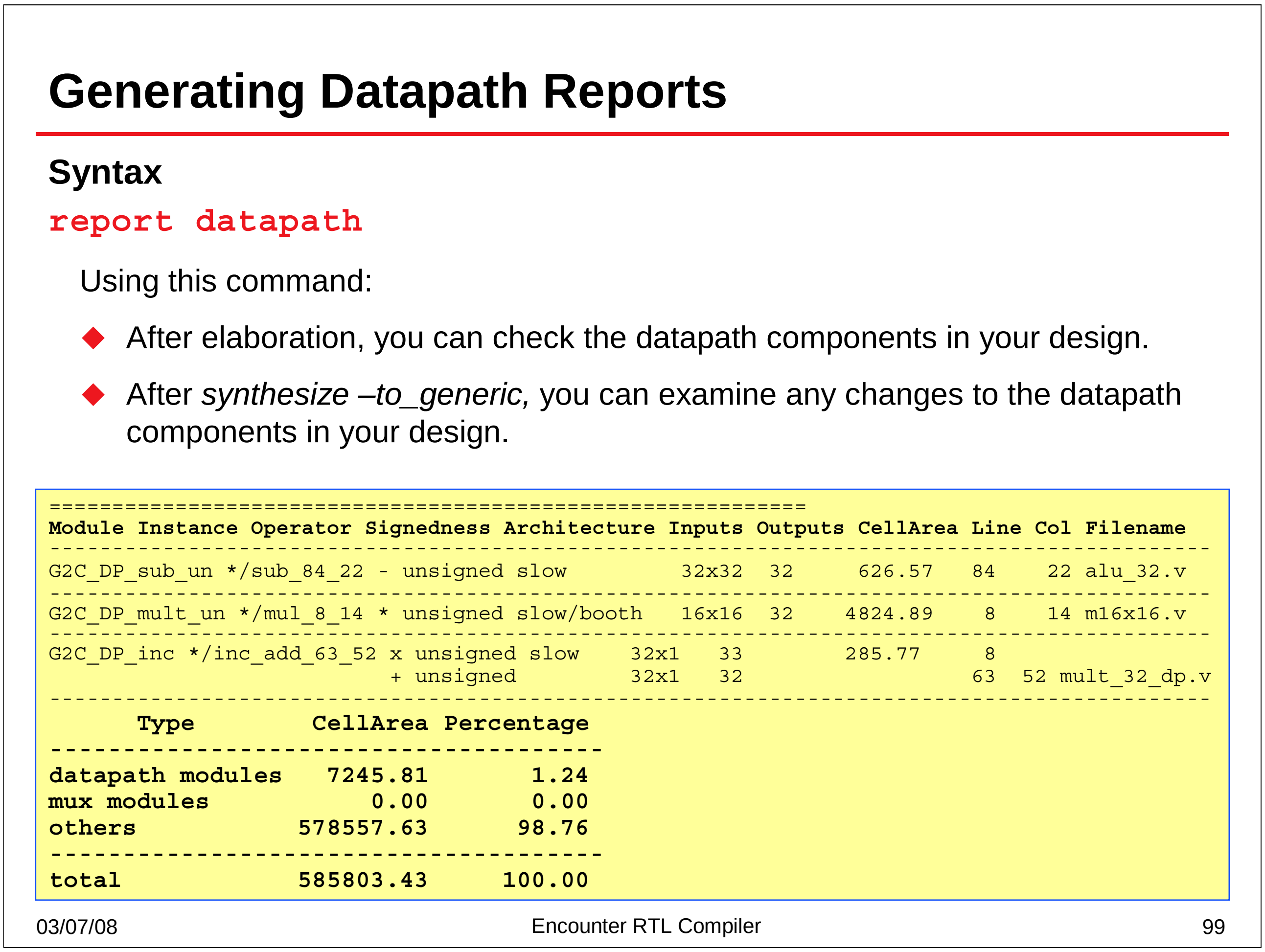
⇨ Recommended, but can be skipped for mapped netlist. If skipped, constraints might need to be modified appropriately.

03/07/08 Encounter RTL Compiler 98

Retiming Registers with Asynchronous Set and Reset Signals

Setting the *retime_async_reset* attribute to true will retime those registers that have either a set or reset signal. Registers that have both set and reset signals will not be retimed in any case.

Optimize registers with reset signals using the *retime_optimize_reset* attribute. Set this attribute to replace registers (that have *dont_care* set or reset condition) using simple flops that don’t have set or reset inputs. This attribute needs to be set in addition to the *retime_async_reset* attribute.



Generating Datapath Reports

Syntax

`report datapath`

Using this command:

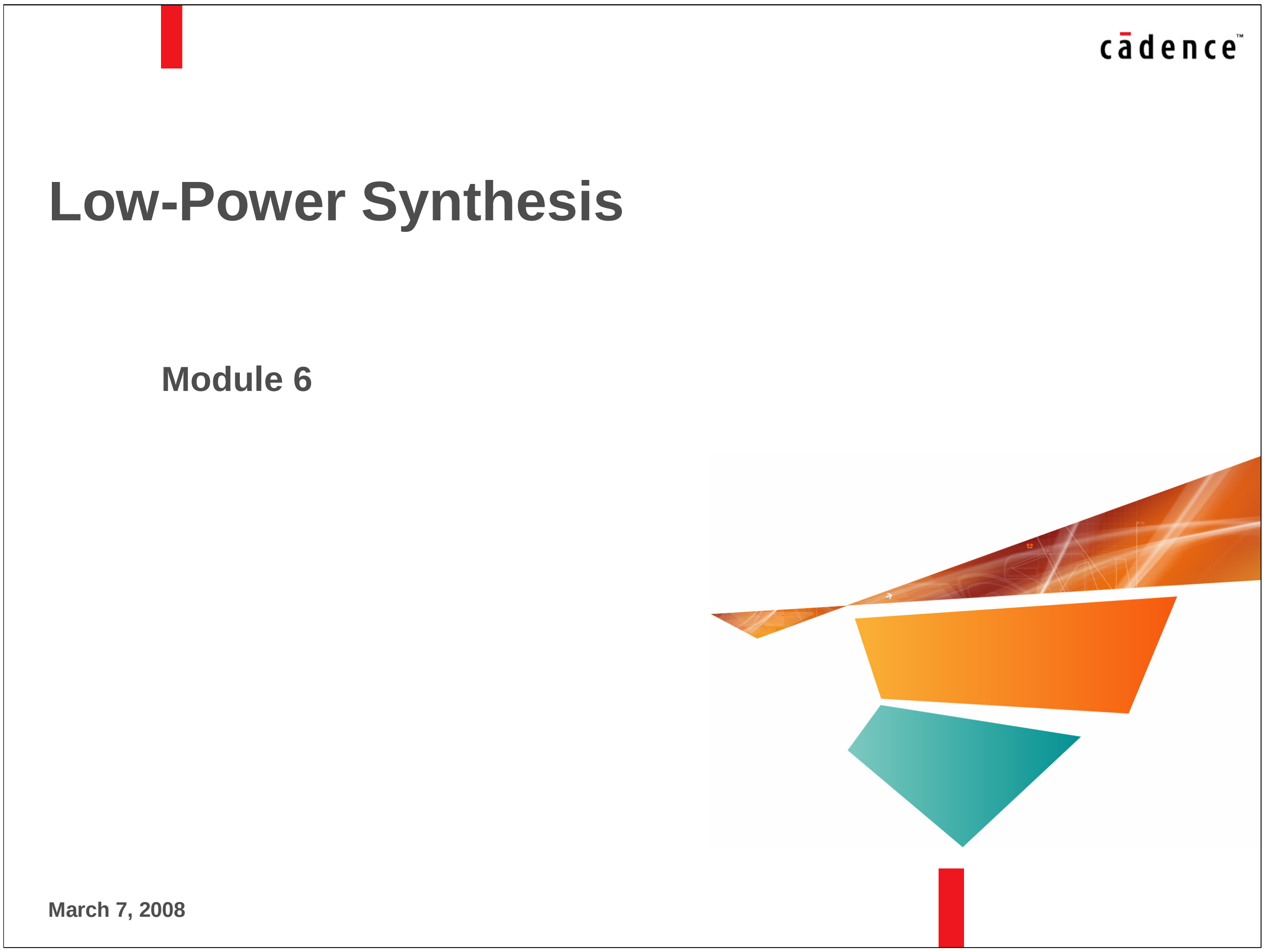
- After elaboration, you can check the datapath components in your design.
- After *synthesize –to_generic,* you can examine any changes to the datapath components in your design.

```
============================================================
Module Instance Operator Signedness Architecture Inputs Outputs CellArea Line Col Filename
-------------------------------------------------------------------------------------------
G2C_DP_sub_un */sub_84_22 - unsigned slow         32x32  32      626.57   84     22 alu_32.v
-------------------------------------------------------------------------------------------
G2C_DP_mult_un */mul_8_14 * unsigned slow/booth   16x16  32     4824.89    8     14 m16x16.v
-------------------------------------------------------------------------------------------
G2C_DP_inc */inc_add_63_52 x unsigned slow    32x1   33         285.77     8
                           + unsigned         32x1   32                   63   52 mult_32_dp.v
-------------------------------------------------------------------------------------------
      Type          CellArea Percentage
--------------------------------------------
datapath modules    7245.81        1.24
mux modules            0.00        0.00
others            578557.63       98.76
--------------------------------------------
total             585803.43      100.00
```

Use the *report datapath* command to:

- Identify datapath operators.
- Examine how carrysave arithmetic (CSA) transformations are applied.
- Examine the selected architectures.
- Examine the datapath area.

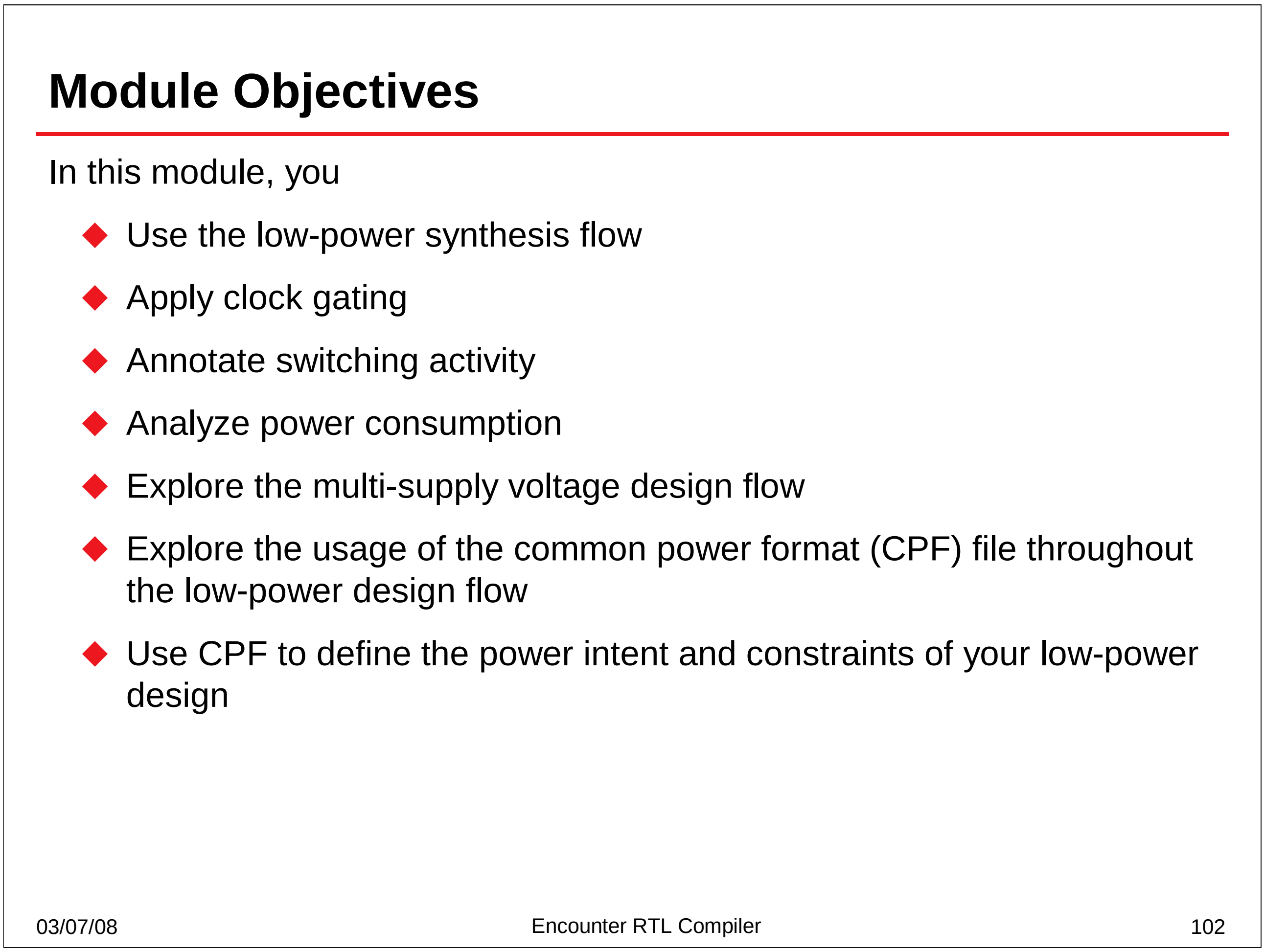
To report the line numbers of the datapath components in the RTL code, set the *hdl_track_filename_row_col* root attribute to *true*.



Low-Power Synthesis

Module 6

March 7, 2008



Module Objectives

In this module, you

- Use the low-power synthesis flow
- Apply clock gating
- Annotate switching activity
- Analyze power consumption
- Explore the multi-supply voltage design flow
- Explore the usage of the common power format (CPF) file throughout the low-power design flow
- Use CPF to define the power intent and constraints of your low-power design

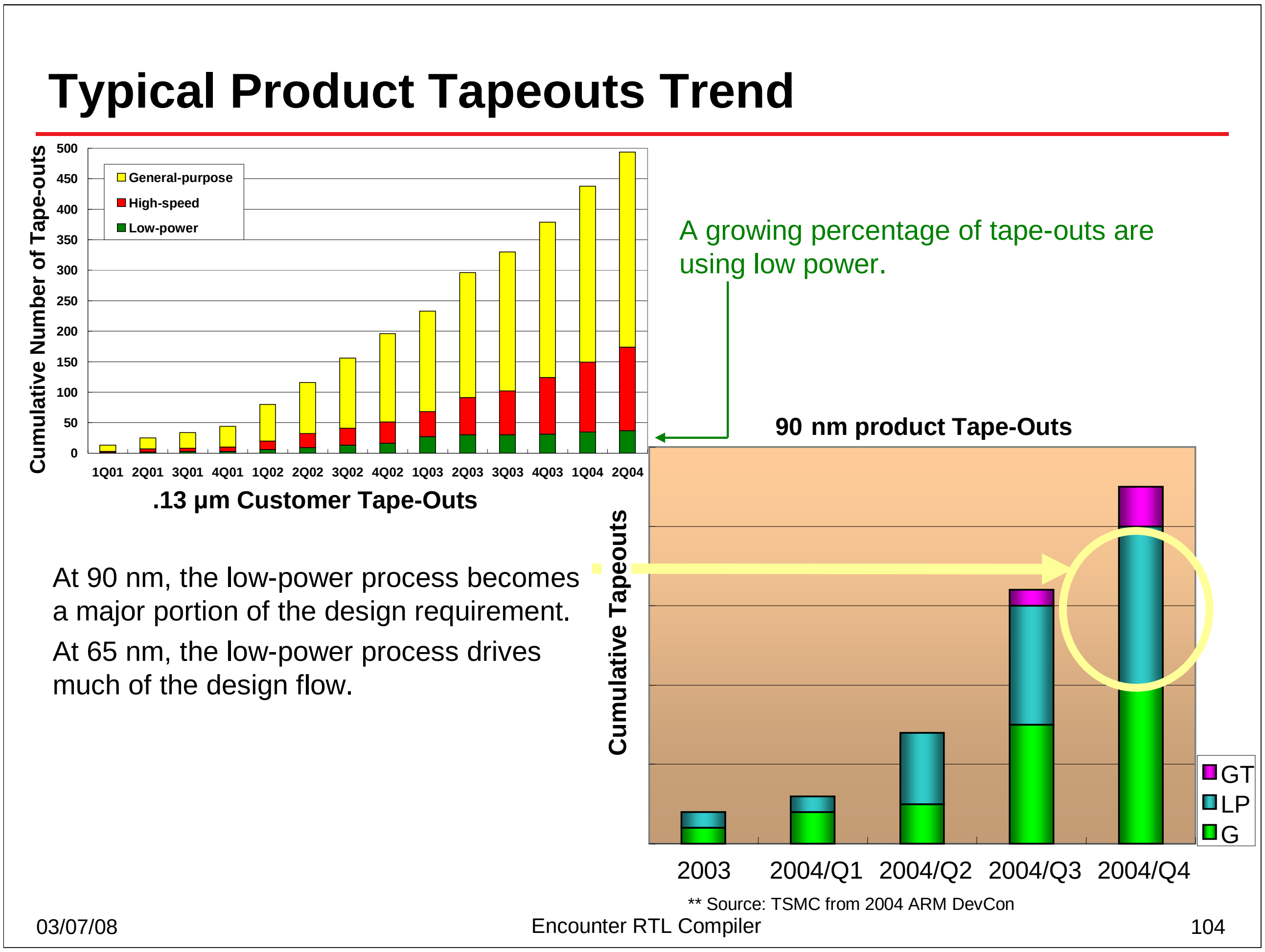
Motivation for Power Management

Longer Battery life between charges

Maximum power consumption is limiting performance

Cost and reliability of devices

Everyone is motivated to do something about power management.



Typical Product Tapeouts Trend

.13 µm Customer Tape-Outs

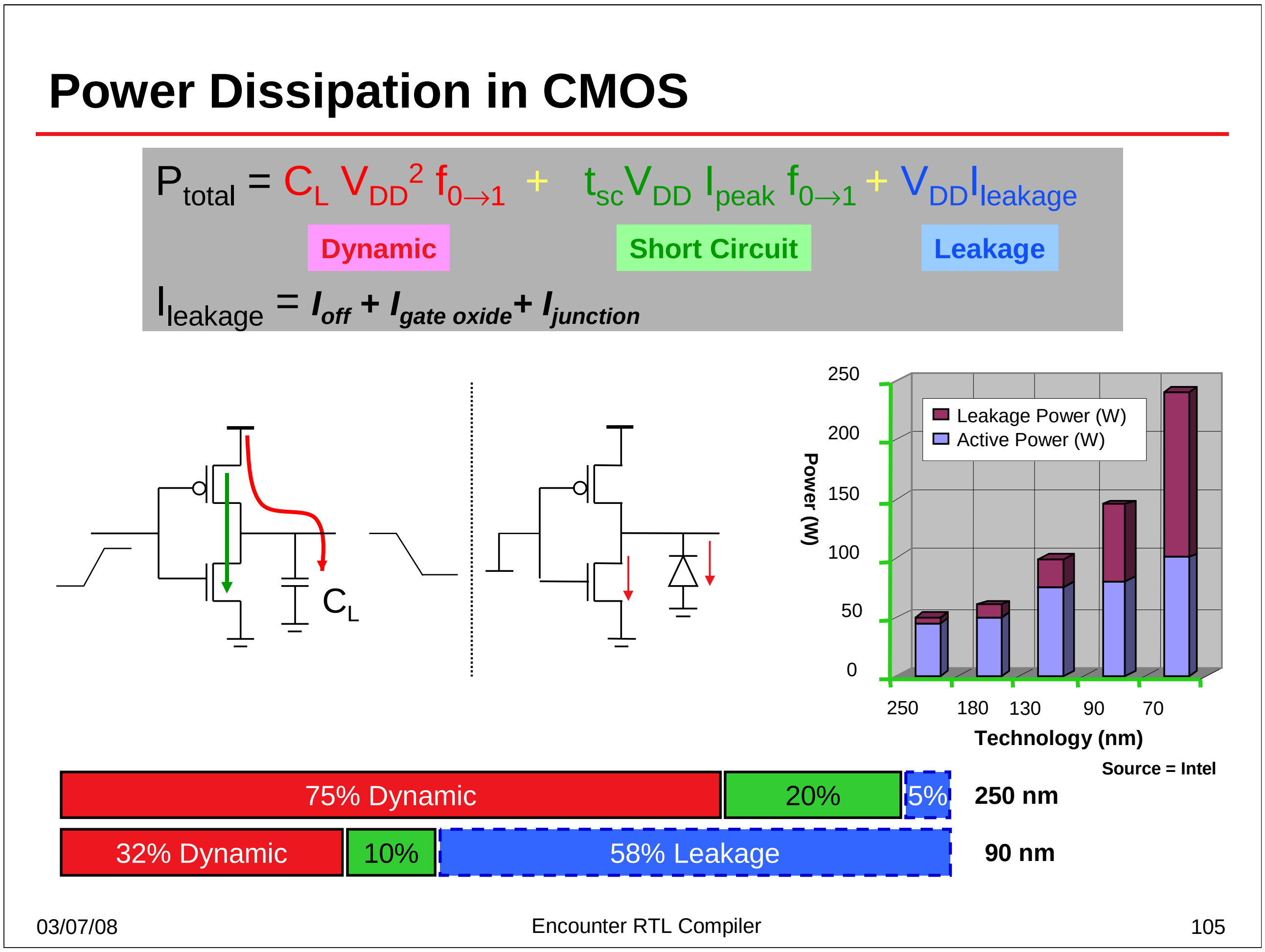
A growing percentage of tape-outs are using low power.

90 nm product Tape-Outs

At 90 nm, the low-power process becomes a major portion of the design requirement.

At 65 nm, the low-power process drives much of the design flow.

** Source: TSMC from 2004 ARM DevCon



Power Dissipation in CMOS

$$P_{total} = C_L V_{DD}^2 f_{0\to1} + t_{sc} V_{DD} I_{peak} f_{0\to1} + V_{DD} I_{leakage}$$

Dynamic

Short Circuit

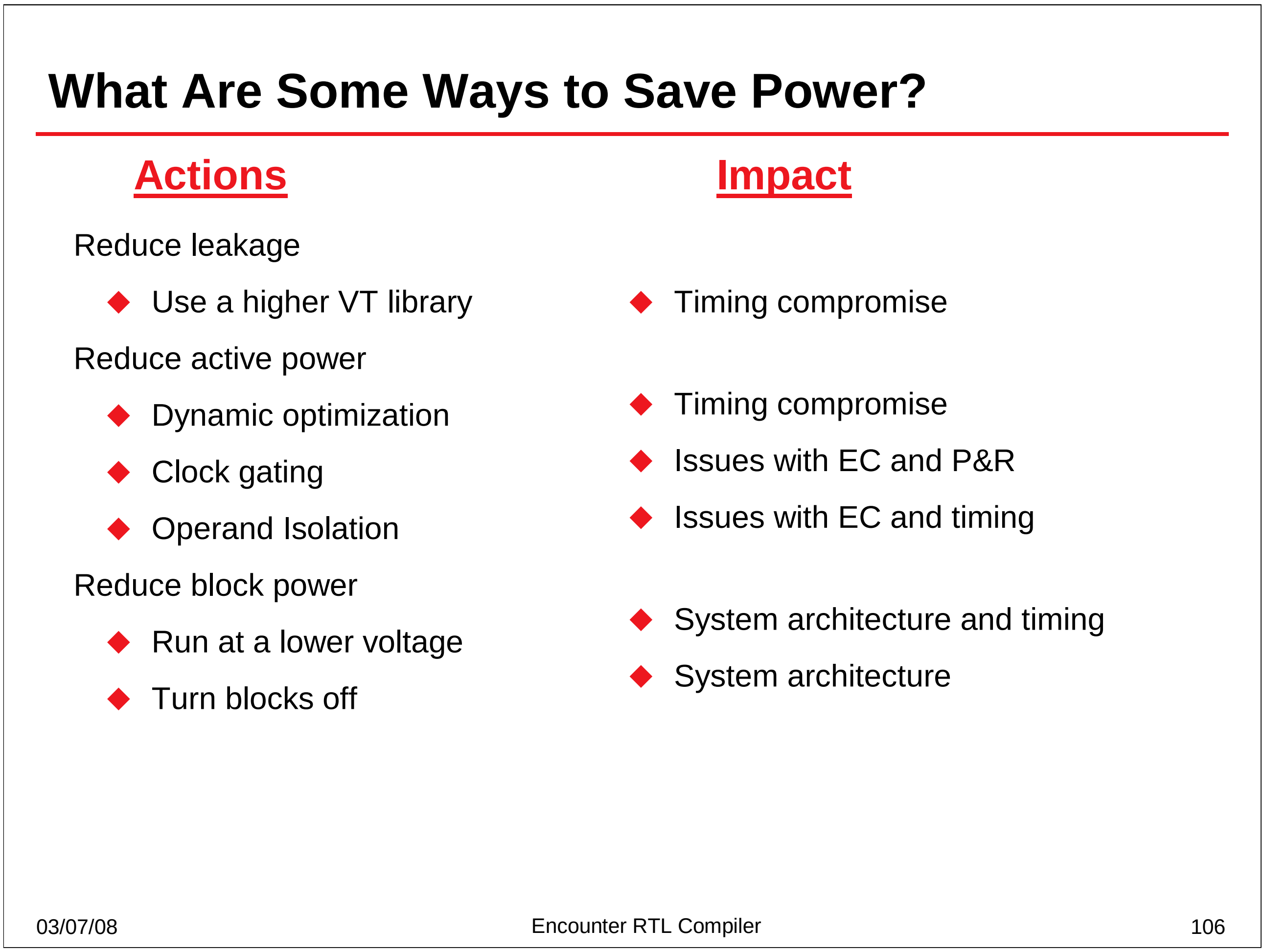
Leakage

$$I_{leakage} = I_{off} + I_{gate\ oxide} + I_{junction}$$

Source = Intel

75% Dynamic	20%	5%	250 nm
32% Dynamic	10%	58% Leakage	90 nm

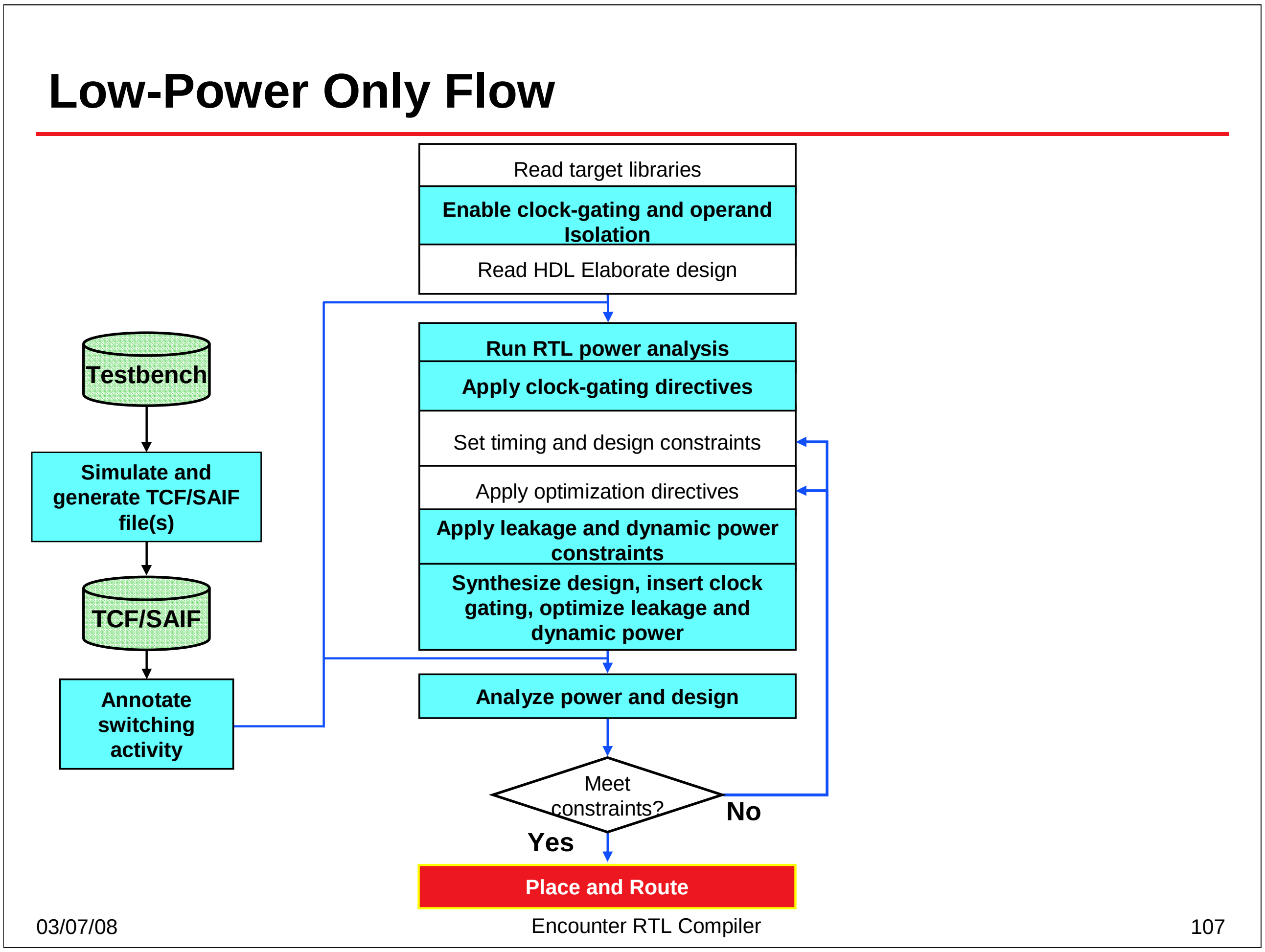
At 250 nm, the leakage power was only 5%, but increased to 58% at 90 nm.



What Are Some Ways to Save Power?

Actions	Impact
Reduce leakage	
◆ Use a higher VT library	◆ Timing compromise
Reduce active power	
◆ Dynamic optimization	◆ Timing compromise
◆ Clock gating	◆ Issues with EC and P&R
◆ Operand Isolation	◆ Issues with EC and timing
Reduce block power	
◆ Run at a lower voltage	◆ System architecture and timing
◆ Turn blocks off	◆ System architecture

EC stands for equivalence checking.



Low-Power Only Flow

Read target libraries

Enable clock-gating and operand Isolation

Read HDL Elaborate design

Run RTL power analysis

Apply clock-gating directives

Set timing and design constraints

Apply optimization directives

Apply leakage and dynamic power constraints

Synthesize design, insert clock gating, optimize leakage and dynamic power

Analyze power and design

Meet constraints?

No

Yes

Place and Route

Testbench

Simulate and generate TCF/SAIF file(s)

TCF/SAIF

Annotate switching activity

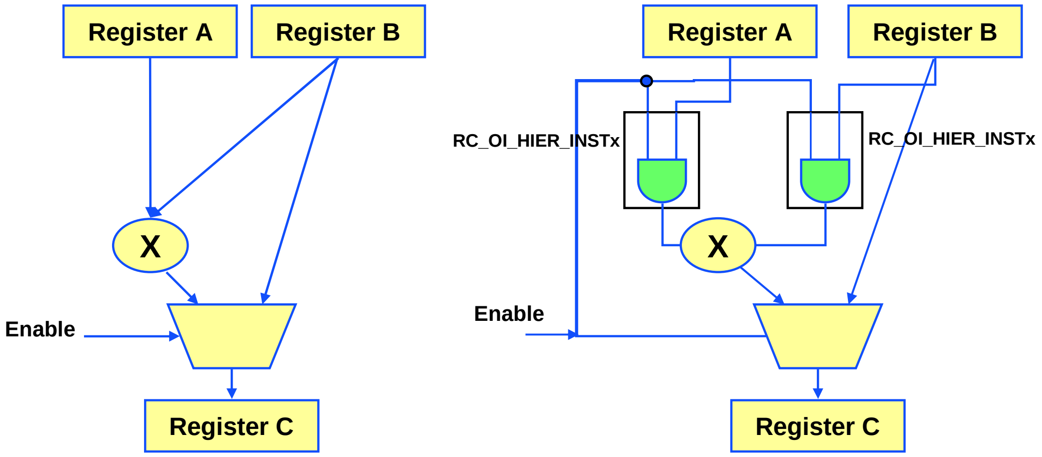
03/07/08 Encounter RTL Compiler 107

Enabling Operand Isolation

To ensure that operand-isolation logic is inserted during synthesis, set the following attribute:

```
set_attribute lp_insert_operand_isolation true /
```

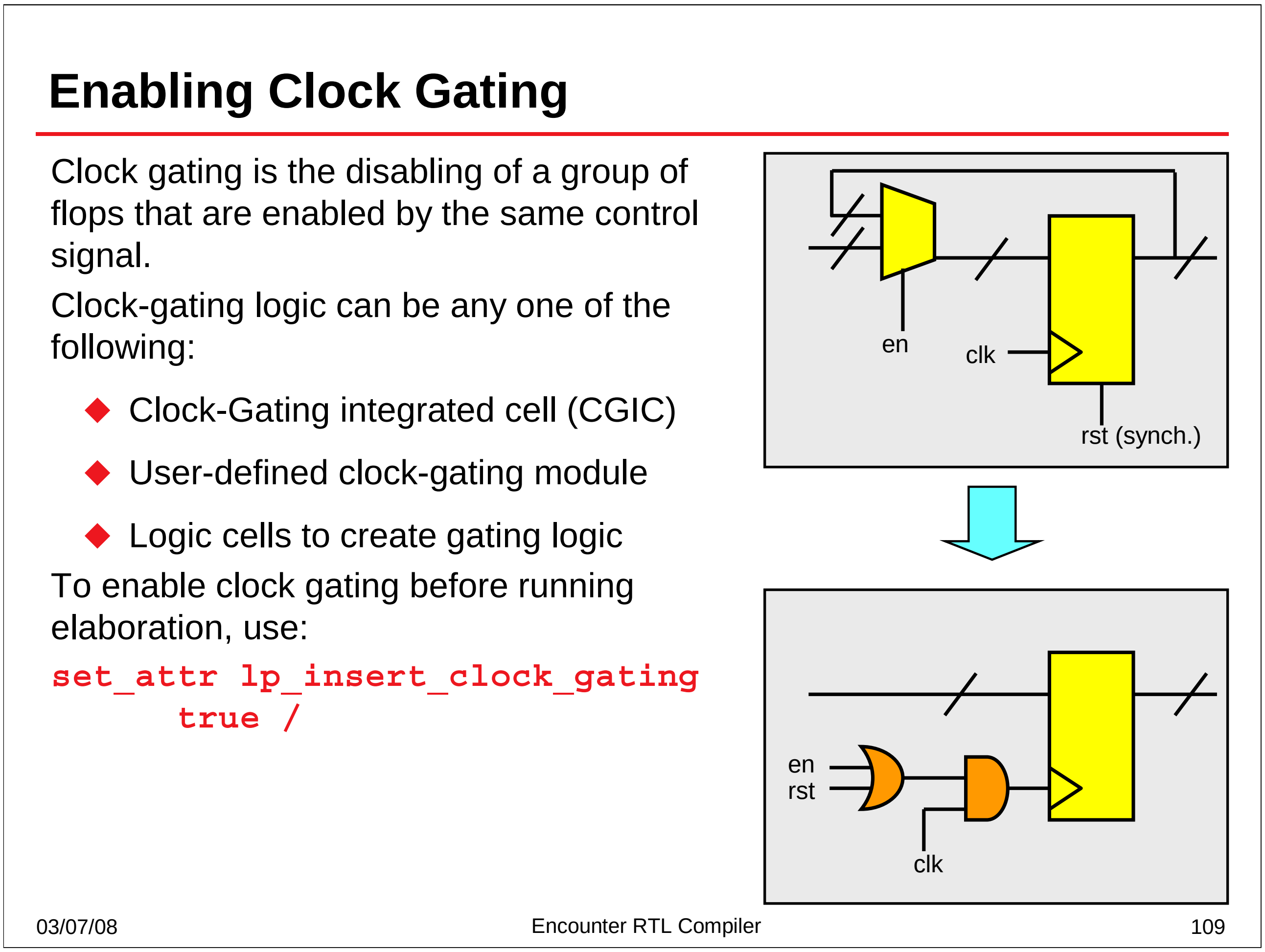
- Specify this attribute before the elaborate command.



Before Operand Isolation

After Operand Isolation

- During elaboration, given the RTL netlist of the design, the low power engine identifies the datapath block candidates (such as adders and multipliers) for operand isolation and inserts operand isolation instances.
- Commitment and decommitment of the operand isolation instances happens during synthesis.
- Only datapath blocks with an input width equal to or larger than 8 are considered.



Enabling Clock Gating

Clock gating is the disabling of a group of flops that are enabled by the same control signal.

Clock-gating logic can be any one of the following:

- Clock-Gating integrated cell (CGIC)
- User-defined clock-gating module
- Logic cells to create gating logic

To enable clock gating before running elaboration, use:

```
set_attr lp_insert_clock_gating
      true /
```

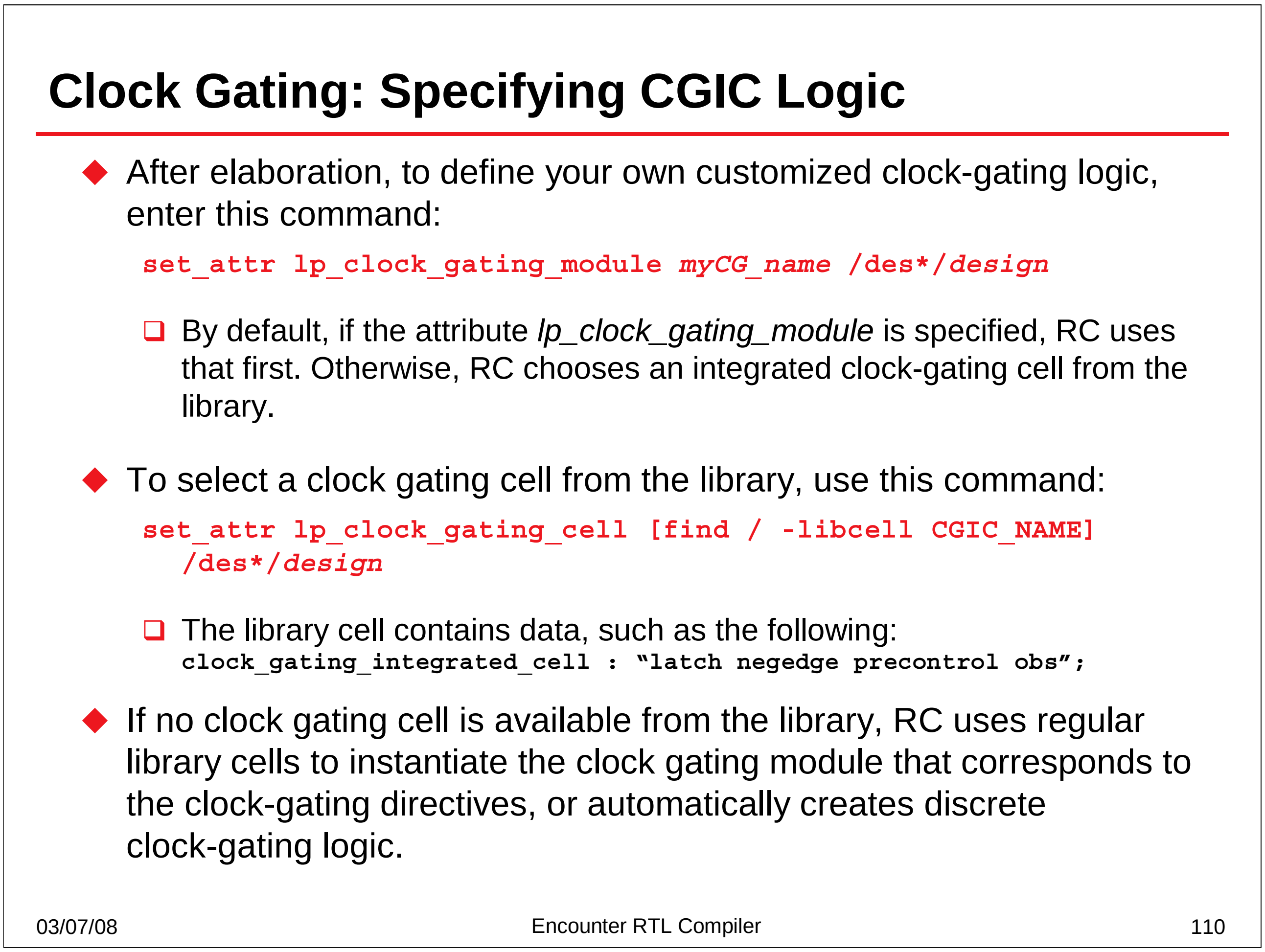
03/07/08 Encounter RTL Compiler 109

In many designs, data is loaded into registers infrequently, but the clock signal continues to switch at every clock cycle, often driving a large capacitive load. You can save a significant amount of power by identifying when the registers are inactive and by disabling the clock during these periods.

Clock gating is the primary form of reducing power consumption (dynamic).

- Clock gating reduces dynamic power consumption caused by switching.
- Clock gating shuts down registers when their inputs do not change.

Clock gating happens during elaboration, identifying registers whose outputs are enabled, and essentially pulling the enable in front of the register. It has the intelligence to identify groups of flops that share the same control signal and use a single gating function for them. Encounter® RTL Complier will even “declone” gating logic with common enable signals across the hierarchy.



Clock Gating: Specifying CGIC Logic

- After elaboration, to define your own customized clock-gating logic, enter this command:

  ```
  set_attr lp_clock_gating_module myCG_name /des*/design
  ```

  - By default, if the attribute *lp_clock_gating_module* is specified, RC uses that first. Otherwise, RC chooses an integrated clock-gating cell from the library.

- To select a clock gating cell from the library, use this command:

  ```
  set_attr lp_clock_gating_cell [find / -libcell CGIC_NAME]
    /des*/design
  ```

  - The library cell contains data, such as the following:

  ```
  clock_gating_integrated_cell : “latch negedge precontrol obs”;
  ```

- If no clock gating cell is available from the library, RC uses regular library cells to instantiate the clock gating module that corresponds to the clock-gating directives, or automatically creates discrete clock-gating logic.

03/07/08 Encounter RTL Compiler 110

CGIC: Clock gating integrated cell.

To specify the type of clock-gating logic the tool must create, use the following attributes:

- `set_attr lp_clock_gating_style {none | latch | ff}`
- `set_attr lp_clock_gating_add_obs_port {true | false}`
- `set_attr lp_clock_gating_add_reset {true | false}`
- `set_attr lp_clock_gating_control_point {none| precontrol| postcontrol}`

To prevent adding clock-gating throughout a design, a subdesign, or a unique hierarchical instance, set the following attribute:

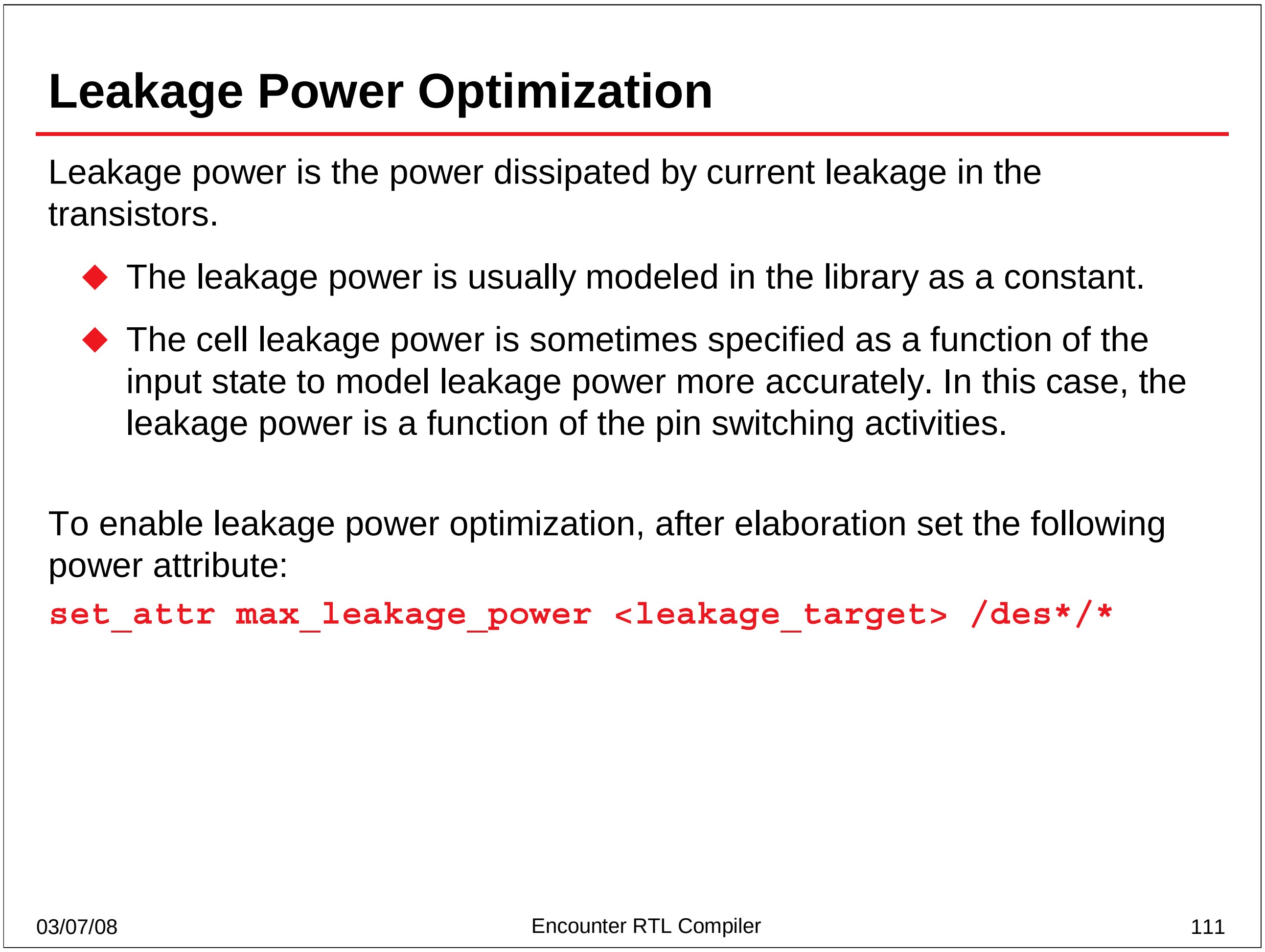
- `set_attr lp_clock_gating_exclude true <object_path>`

To specify the minimum number of registers required to enable clock-gating insertion, set the following root-level attribute:

- `set_attr lp_clock_gating_min_flops integer /`
- Default: 3

To specify the maximum number of registers that can be driven by each clock-gating element, set the following root-level attribute:

- `set_attr lp_clock_gating_max_flops integer /`
- Default: 2048



Leakage Power Optimization

Leakage power is the power dissipated by current leakage in the transistors.

- The leakage power is usually modeled in the library as a constant.
- The cell leakage power is sometimes specified as a function of the input state to model leakage power more accurately. In this case, the leakage power is a function of the pin switching activities.

To enable leakage power optimization, after elaboration set the following power attribute:

```
set_attr max_leakage_power <leakage_target> /des*/*
```

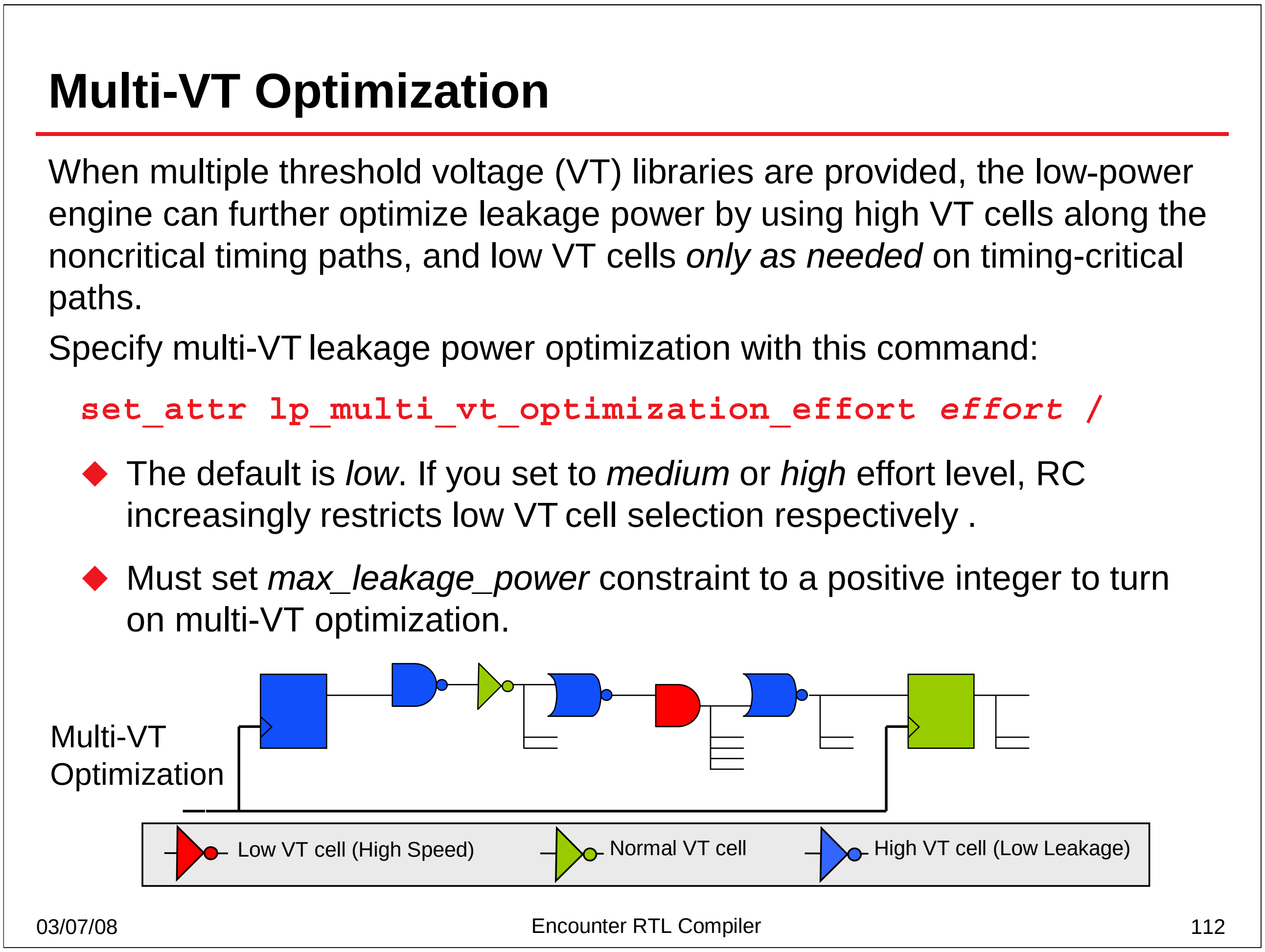
03/07/08 Encounter RTL Compiler 111

VT: Voltage Threshold (V_{th})

If the library uses a constant leakage power model, the leakage power optimization is entirely determined by the cell leakage power specified in the technology library. The cell leakage power can be determined by the following *.lib* attributes:

- `cell_leakage_power`
- `default_cell_leakage_power`
- `default_leakage_power_density`
- `leakage_power_unit`
- `k_process_cell_leakage_power`
- `k_temp_cell_leakage_power`
- `k_volt_cell_leakage_power`

If the library uses a state-dependent leakage power model (modeled using the leakage_power group in the cell group), you also need to read in switching activities before optimization to increase the power estimation accuracy.



Multi-VT Optimization

When multiple threshold voltage (VT) libraries are provided, the low-power engine can further optimize leakage power by using high VT cells along the noncritical timing paths, and low VT cells *only as needed* on timing-critical paths.

Specify multi-VT leakage power optimization with this command:

```
set_attr lp_multi_vt_optimization_effort effort /
```

- The default is *low*. If you set to *medium* or *high* effort level, RC increasingly restricts low VT cell selection respectively .
- Must set *max_leakage_power* constraint to a positive integer to turn on multi-VT optimization.

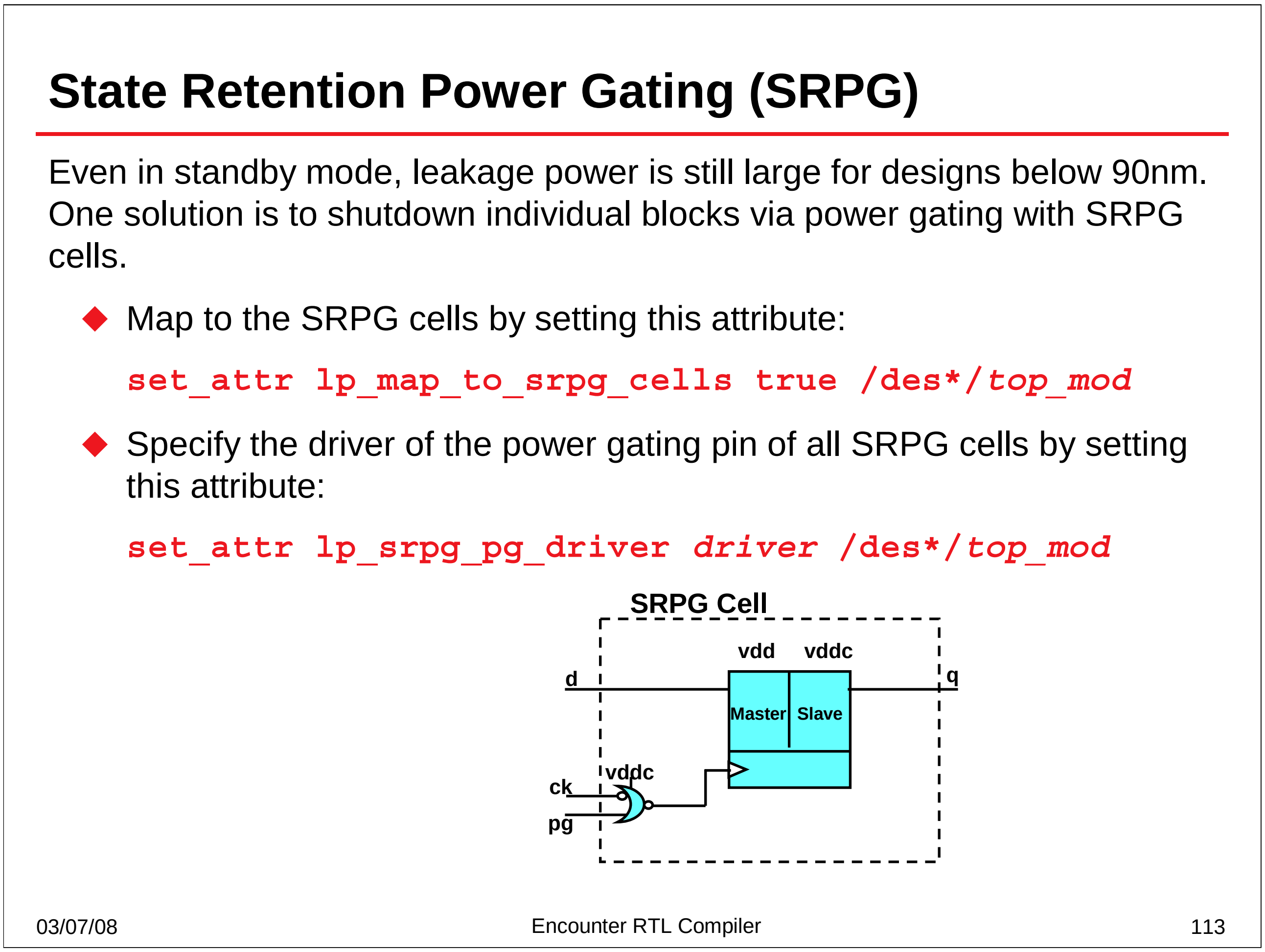
03/07/08 Encounter RTL Compiler 112

Leakage power can be reduced by using high VT library cells wherever possible.

- You must read all the multi-VT libraries initially for RC to make optimal trade-offs during synthesis.
- High VT cells have low leakage power and slow timing performance and are good along the noncritical timing paths.
- Low VT cells have higher leakage power and faster timing performance and are good for timing-critical paths.

```
set_attr library {low_vt.lib, std_vt.lib, high_vt.lib}
set_attr max_leakage_power <leakage_target> /design/*
set_attr lp_multi_vt_optimization_effort <low/med/high>
```

- **low**: (default) Uses low VT cells on timing critical paths, and high VT cells on less critical paths
- **med**: restricts the use of low VT cells in order to focus more on leakage
- **high**: further restricts the use of low VT cells



State Retention Power Gating (SRPG)

Even in standby mode, leakage power is still large for designs below 90nm. One solution is to shutdown individual blocks via power gating with SRPG cells.

- Map to the SRPG cells by setting this attribute:

```
set_attr lp_map_to_srpg_cells true /des*/top_mod
```

- Specify the driver of the power gating pin of all SRPG cells by setting this attribute:

```
set_attr lp_srpg_pg_driver driver /des*/top_mod
```

03/07/08 Encounter RTL Compiler 113

State Retention Register Switching

Power Down Sequence:

1. System indication of deep sleep mode
2. Clocks stopped and returned to zero.
3. PG goes high
4. VDD shuts off.
5. VDDC is lowered to retention voltage.

Power Up Sequence:

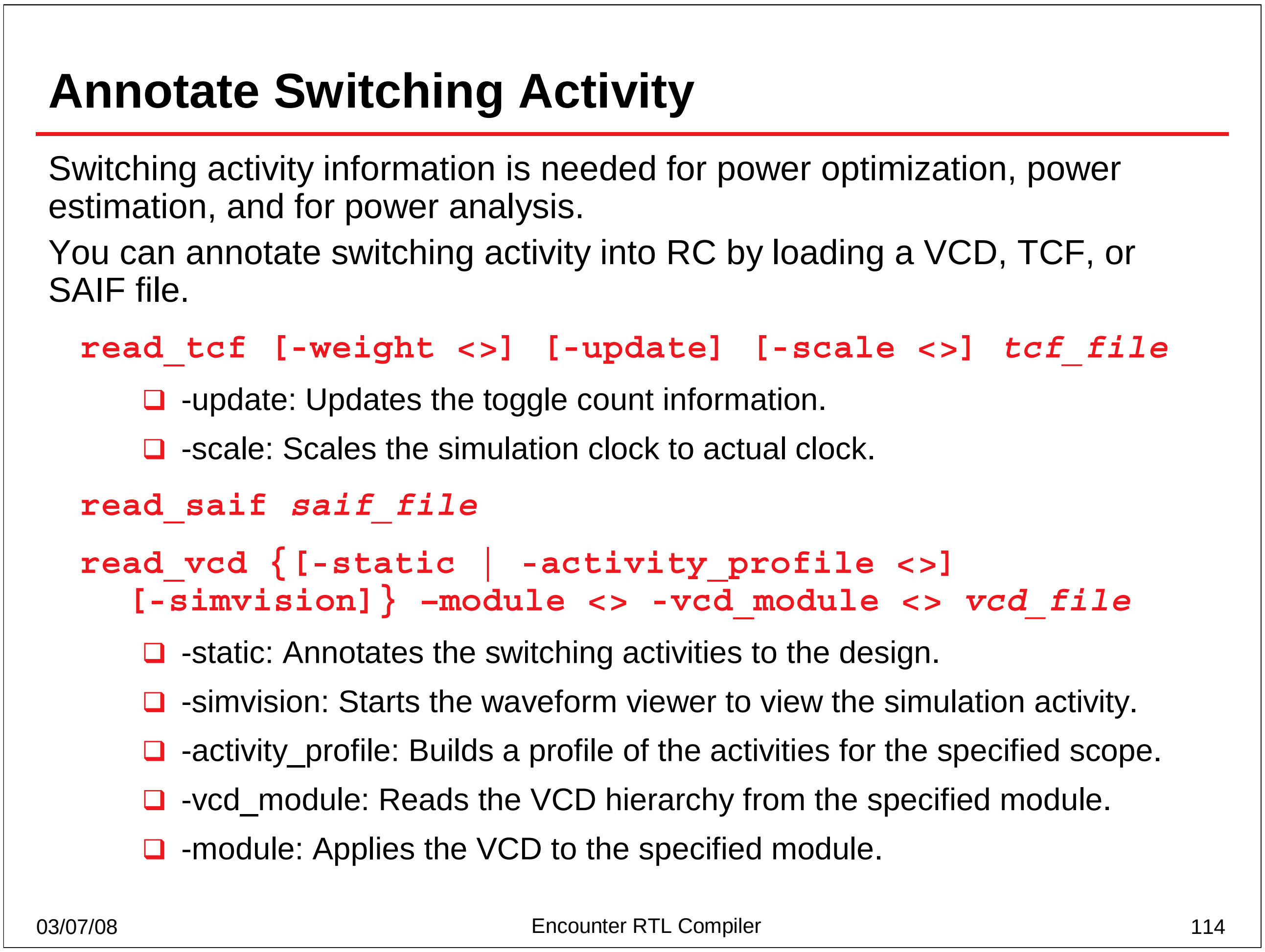
1. System indicator of wake up.
2. VDDC raised to full supply voltage.
3. VDD is turned on.
4. After VDD reaches full value, PG goes low.
5. Clocks enabled.

State Options:

1. Throw away.
2. Scan out to memory.
3. Retain locally in “retention” register.

SRPG Attributes

- **power_gating_cell:** Indicates if a lib cell is SRPG or not. Default is false.
- **power_gating_pin:** Indicates if a lib pin is (SRPG) power gating pin or not. Default is false.
- **power_gating_pin_phase:** Shows the phase of power gating pin. 'true'('false') means it is active high(low). Default is true.



Annotate Switching Activity

Switching activity information is needed for power optimization, power estimation, and for power analysis.

You can annotate switching activity into RC by loading a VCD, TCF, or SAIF file.

```
read_tcf [-weight <>] [-update] [-scale <>] tcf_file
```

- ❑ -update: Updates the toggle count information.
- ❑ -scale: Scales the simulation clock to actual clock.

```
read_saif saif_file
```

```
read_vcd {[-static | -activity_profile <>]
   [-simvision]} -module <> -vcd_module <> vcd_file
```

- ❑ -static: Annotates the switching activities to the design.
- ❑ -simvision: Starts the waveform viewer to view the simulation activity.
- ❑ -activity_profile: Builds a profile of the activities for the specified scope.
- ❑ -vcd_module: Reads the VCD hierarchy from the specified module.
- ❑ -module: Applies the VCD to the specified module.

03/07/08 Encounter RTL Compiler 114

The value change dump (VCD) file contains detailed switching activity from a simulation.

The toggle count format (TCF) file contains switching activity in the form of the toggle count information and the probability of the net or pin to be in the logic 1 state.

The switching activity interchange format (SAIF) file provides detailed information about the switching behavior of nets and ports. This information leads to more accurate power estimation.

To find the source of the probability information, use the following command:

```
get_att lp_probability_type /designs/design/*/nets/net
```

To find the source of the toggle rate information, use this command:

```
get_att lp_toggle_rate_type /designs/design/*/nets/net
```

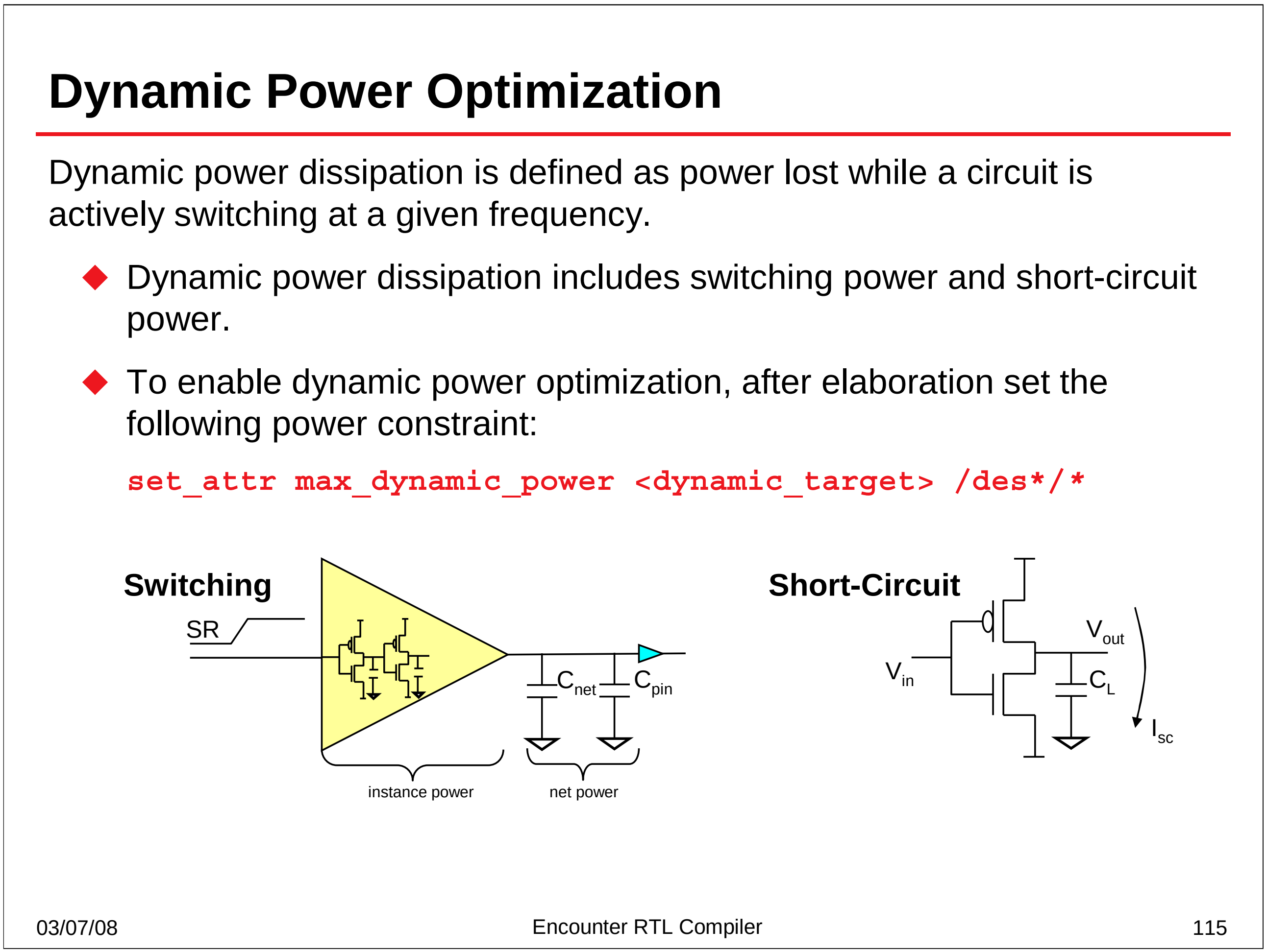
To skip simulation, you can set the net switching activities using the following commands:

```
set_attribute lp_asserted_probability /designs/design/*/nets/net value
set_attribute lp_asserted_toggle_rate /designs/design/*/nets/net value
```

Nets with no switching activity use a default signal probability of 0.5 and a default transition density of 0.02 per nanosecond. You can change these defaults using the following commands:

```
set_attribute lp_default_probability /designs/design value
set_attribute lp_default_toggle_rate /designs/design value
```

Add instances to the scope of activity profiling by setting the *lp_dynamic_analysis_scope* attribute on the instance to *true*.



Dynamic Power Optimization

Dynamic power dissipation is defined as power lost while a circuit is actively switching at a given frequency.

- Dynamic power dissipation includes switching power and short-circuit power.
- To enable dynamic power optimization, after elaboration set the following power constraint:

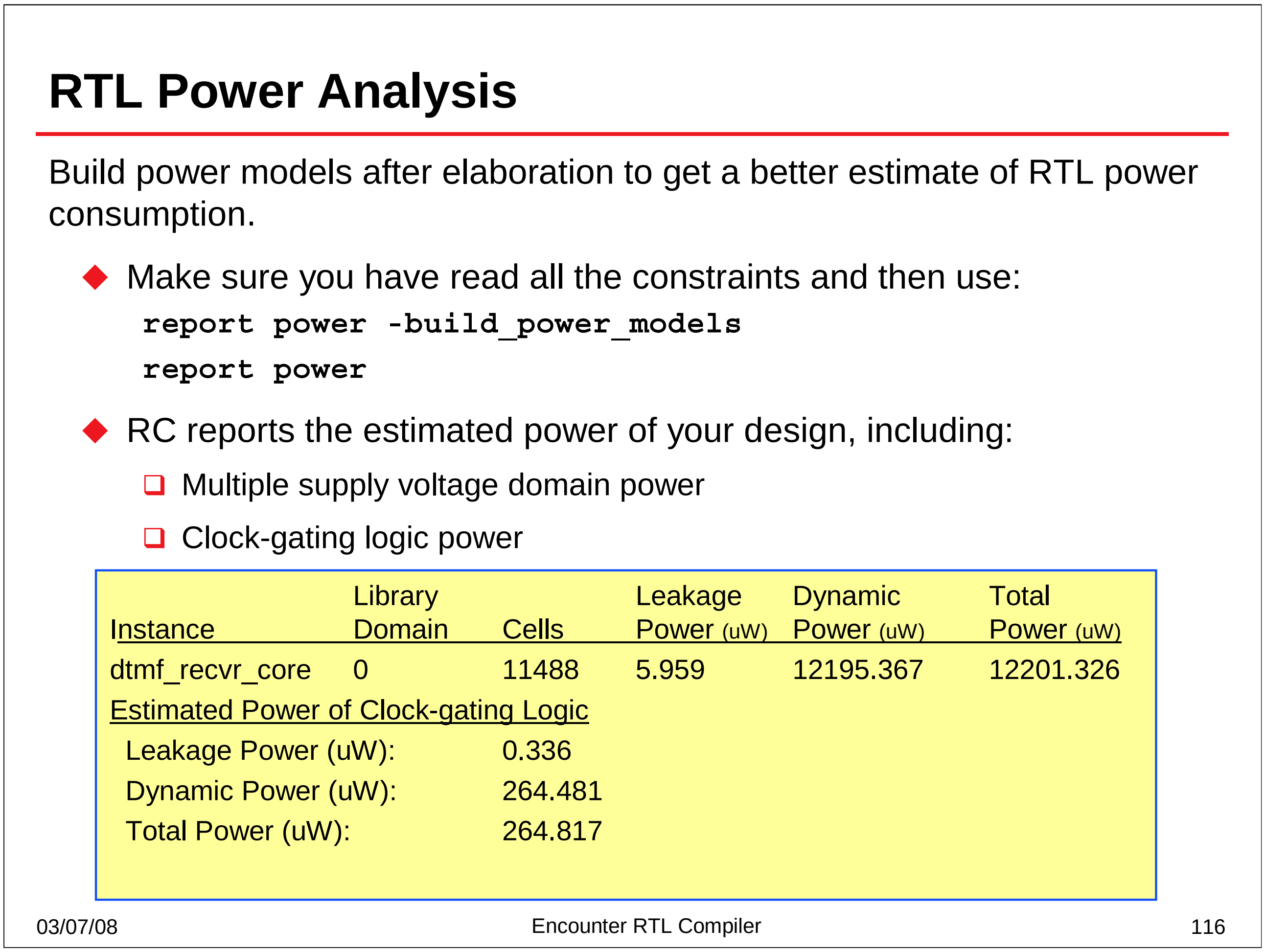
```
set_attr max_dynamic_power <dynamic_target> /des*/*
```

To let the low-power engine optimize the leakage and dynamic power simultaneously, set the following design attribute:

```
set_attr lp_power_optimization_weight weight /des*/design
```

To reduce the dynamic power before optimizing the leakage power, set the following design attribute:

```
set_attr lp_optimize_dynamic_power_first true /des*/design
```



RTL Power Analysis

Build power models after elaboration to get a better estimate of RTL power consumption.

- Make sure you have read all the constraints and then use:

  ```
  report power -build_power_models
  report power
  ```

- RC reports the estimated power of your design, including:
  - Multiple supply voltage domain power
  - Clock-gating logic power

```
                   Library                  Leakage        Dynamic        Total
Instance           Domain      Cells        Power (uW)     Power (uW)     Power (uW)
dtmf_recvr_core    0           11488        5.959          12195.367      12201.326
Estimated Power of Clock-gating Logic
  Leakage Power (uW):          0.336
  Dynamic Power (uW):          264.481
  Total Power (uW):            264.817
```

For accurate RTL power analysis correlation with the final netlist, provide switching activity before building power models.

To enable RTL power analysis, you need to make sure that the following attribute is set:

```
set_attr hdl_track_filename_row_col true /
```

Power Optimization

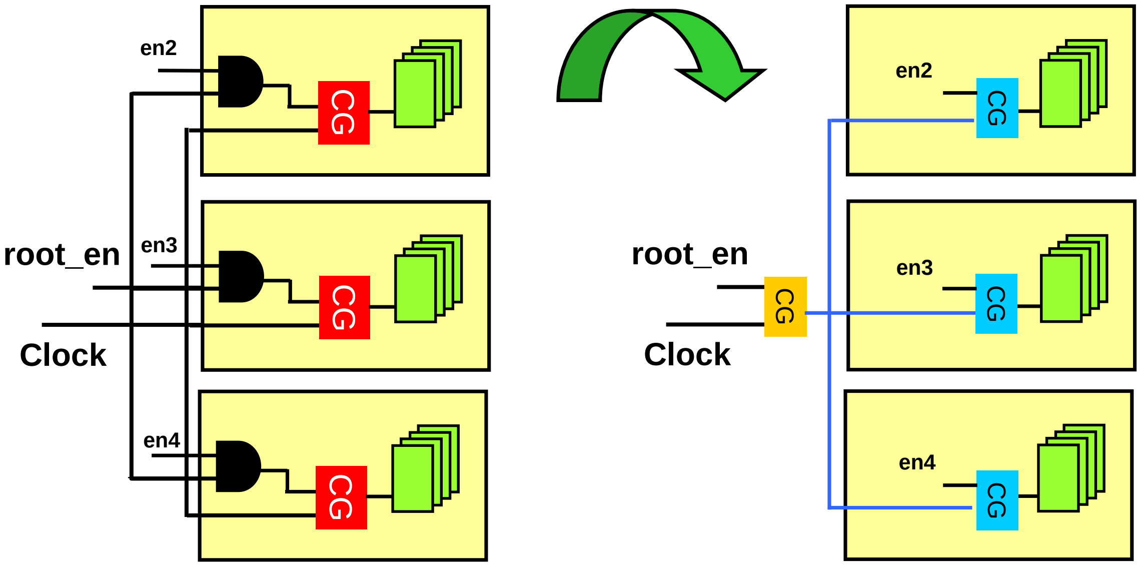
RTL Compiler performs timing and leakage power optimization simultaneously during mapping and incremental optimization as shown in an extract of the log file below:

```
Global mapping status
=====================
                    Worst
                    Total Neg                     Leakage   Switching
Operation           Area      Slack               Power     Power
-------------------------------------------------------------------------------
Path: coeff[3] --> regs/big_normal/data_out_reg[3]/D
power_map           2326      -158                7997      618910
...
Incremental optimization status
===============================
                    Worst     Total - - DRC Totals - -
                    Total Neg Neg       Max       Max       Leakage   Switching
Operation Area      Slack     Slack     Trans     Cap       Power     Power
-------------------------------------------------------------------------------
...
init_power 2560     0         0         0         0         10104     617203
p_rem_buf  2551     0         0         0         0         10047     616922
p_rem_inv  2525     0         0         0         0         10002     616379
p_merge_bi 2506     0         0         0         0         9982      613260
glob_power 2477     0         0         0         0         9387      611549
power_down 2453     0         0         0         0         9087      611250
```

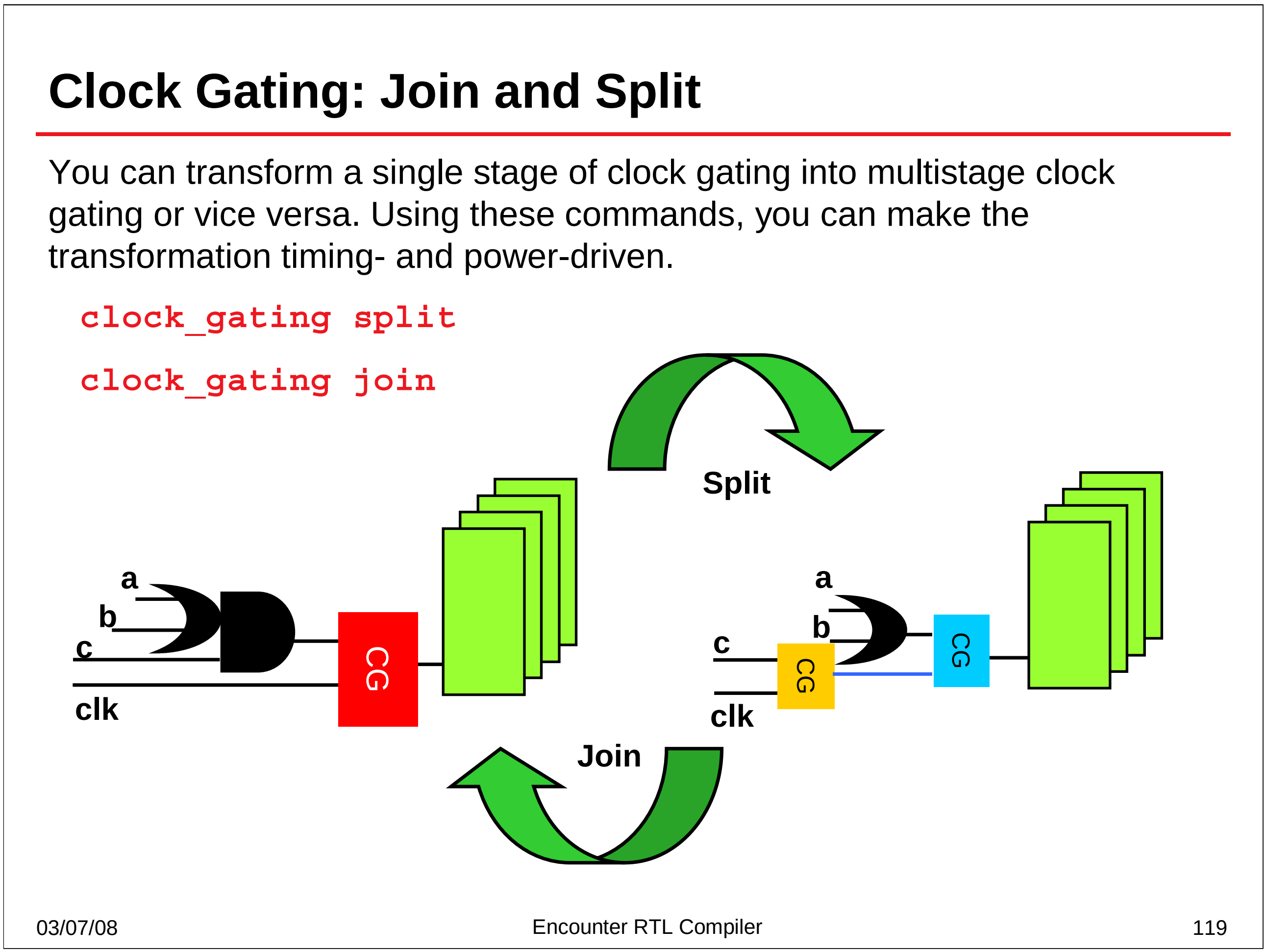
Clock Gating: Share

Extract common subfunctions for the enable logic of clock gating at the leaf level to form multilevel clock gating, by using this command:

```
clock_gating share
```



CG is a clock-gating hierarchical instance. It can contain all glitch removal and design-for-testability (DFT) circuitry.



Clock Gating: Join and Split

You can transform a single stage of clock gating into multistage clock gating or vice versa. Using these commands, you can make the transformation timing- and power-driven.

```
clock_gating split
clock_gating join
```

03/07/08 Encounter RTL Compiler 119

clock_gating split

The split clock-gating logic is a clone of the leaf-level clock-gating logic. This command has these features:

- Helps timing closure if the leaf-level enable is timing critical.

  - Moves faster enable signal closer to the inputs.

  - Enables logic of leaf-level clock gating.

- Improves clock-tree power savings if the leaf-level clock gating has a large fanout.

  The sublevel and root-level clock gating will reduce switching activities of the subclock tree.

clock_gating join

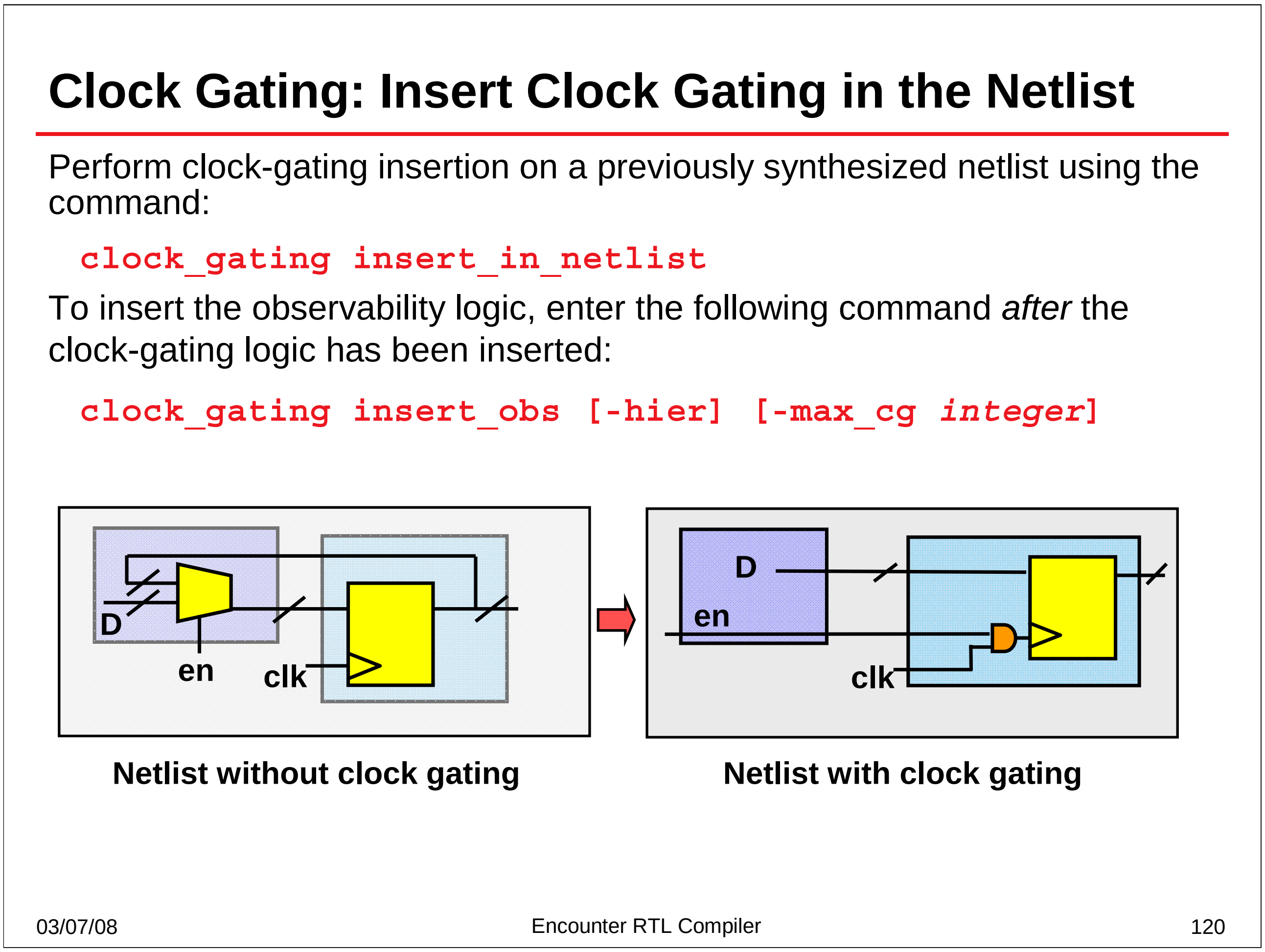
The to-be-merged or joining clock-gating logic must be the same type.

- Helps reduce the area if the design is not timing critical.

  - Reduces the amount of clock gating logic inserted.

  - Simplifies clock tree topology.

- Converts manually instantiated multistage clock gating into simple leaf-level clock-gating logic.



Clock Gating: Insert Clock Gating in the Netlist

Perform clock-gating insertion on a previously synthesized netlist using the command:

```
clock_gating insert_in_netlist
```

To insert the observability logic, enter the following command *after* the clock-gating logic has been inserted:

```
clock_gating insert_obs [-hier] [-max_cg integer]
```

D
en
clk
D
en
clk

Netlist without clock gating **Netlist with clock gating**

Clock gating insertion works only on 2-input MUXes with a feedback loop. Buffers and an even-number of inverters are acceptable along the feedback loop.

The technology library can have up to 26 different types of integrated clock-gating cells. The low power engine supports all 26 types.

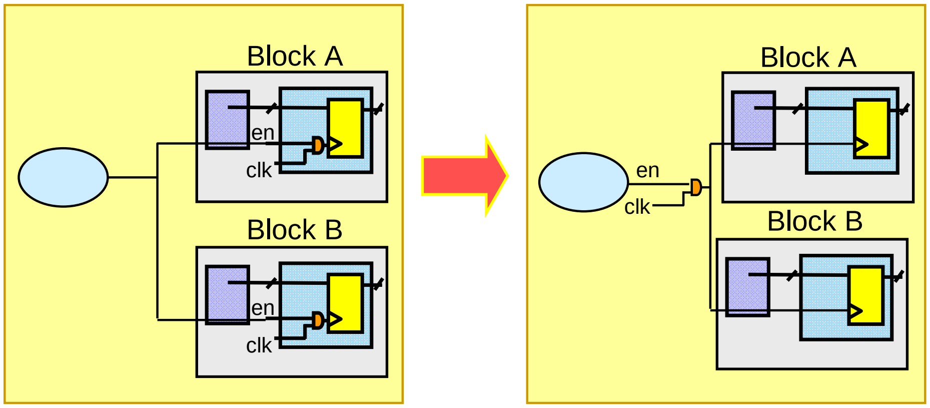
Clock gating is performed on cells with:

- Synchronous set/reset
- Scan enable

Clock Gating: Declone

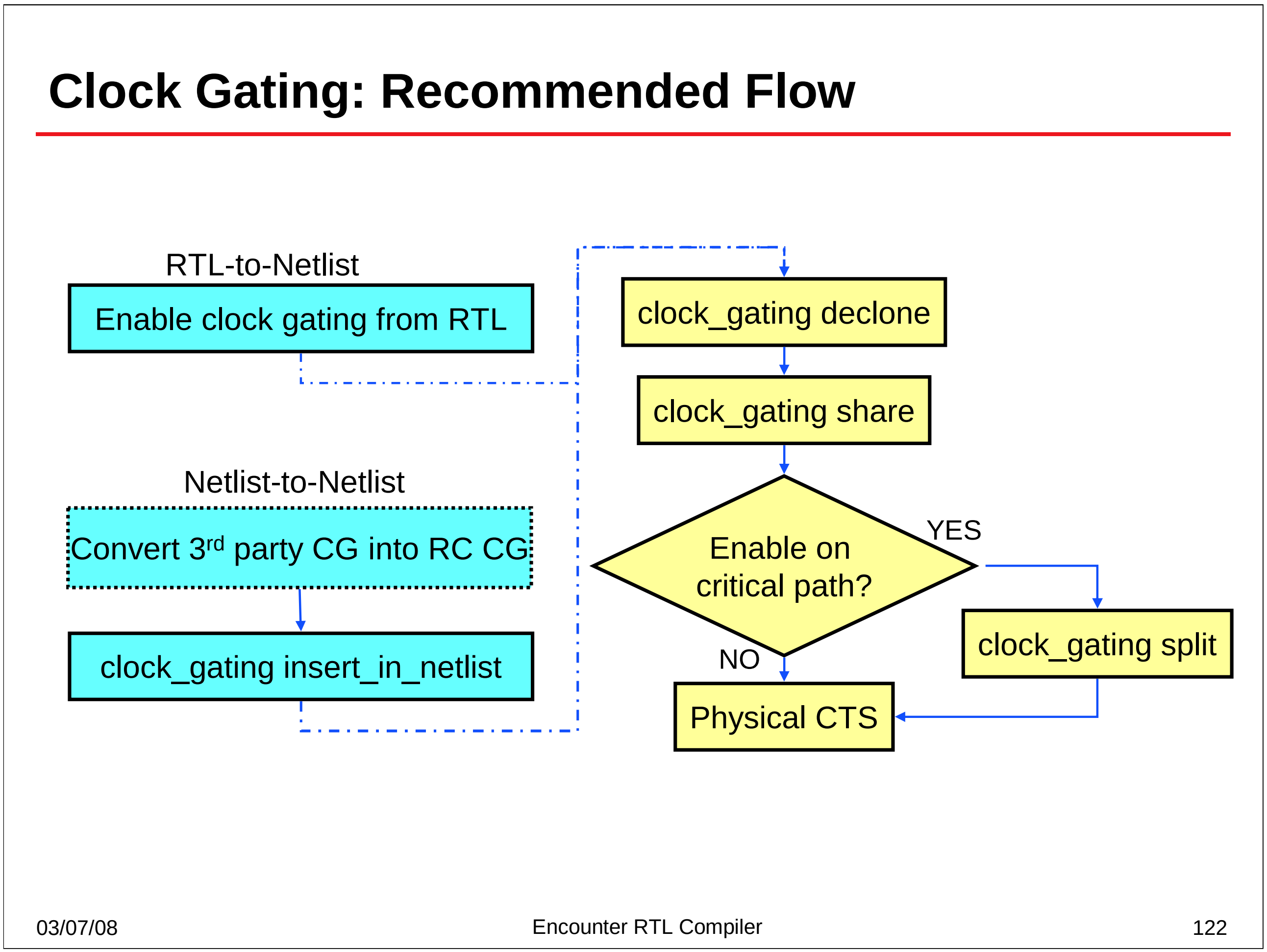
To merge (declone) clock-gating instances across hierarchies, use the following command:

```
clock_gating declone [-hierarchical]
```



To remove the clock-gating instances, use this command:

```
clock_gating remove [-hier | -cg_list inst_list]
```



Clock Gating: Recommended Flow

RTL-to-Netlist

Enable clock gating from RTL

Netlist-to-Netlist

Convert 3rd party CG into RC CG

clock_gating insert_in_netlist

clock_gating declone

clock_gating share

Enable on critical path?

YES

clock_gating split

NO

Physical CTS

03/07/08 Encounter RTL Compiler 122

CTS: Clock tree synthesis

Clock Gating: Generating a Report

To generate a clock-gating report, use the following command:

```
report clock_gating -summary
```

```
Clock Gating Summary
*************************************************
|        Category         | Number | Percentage |
-------------------------------------------------
| Clock Gating Instances |      2 |          - |
-------------------------------------------------
| Gated Flip-flops        |     16 |      100.0 |
| Ungated Flip-flops      |      0 |        0.0 |
=================================================
| Total Flip-flops        |     16 |      100.0 |
```

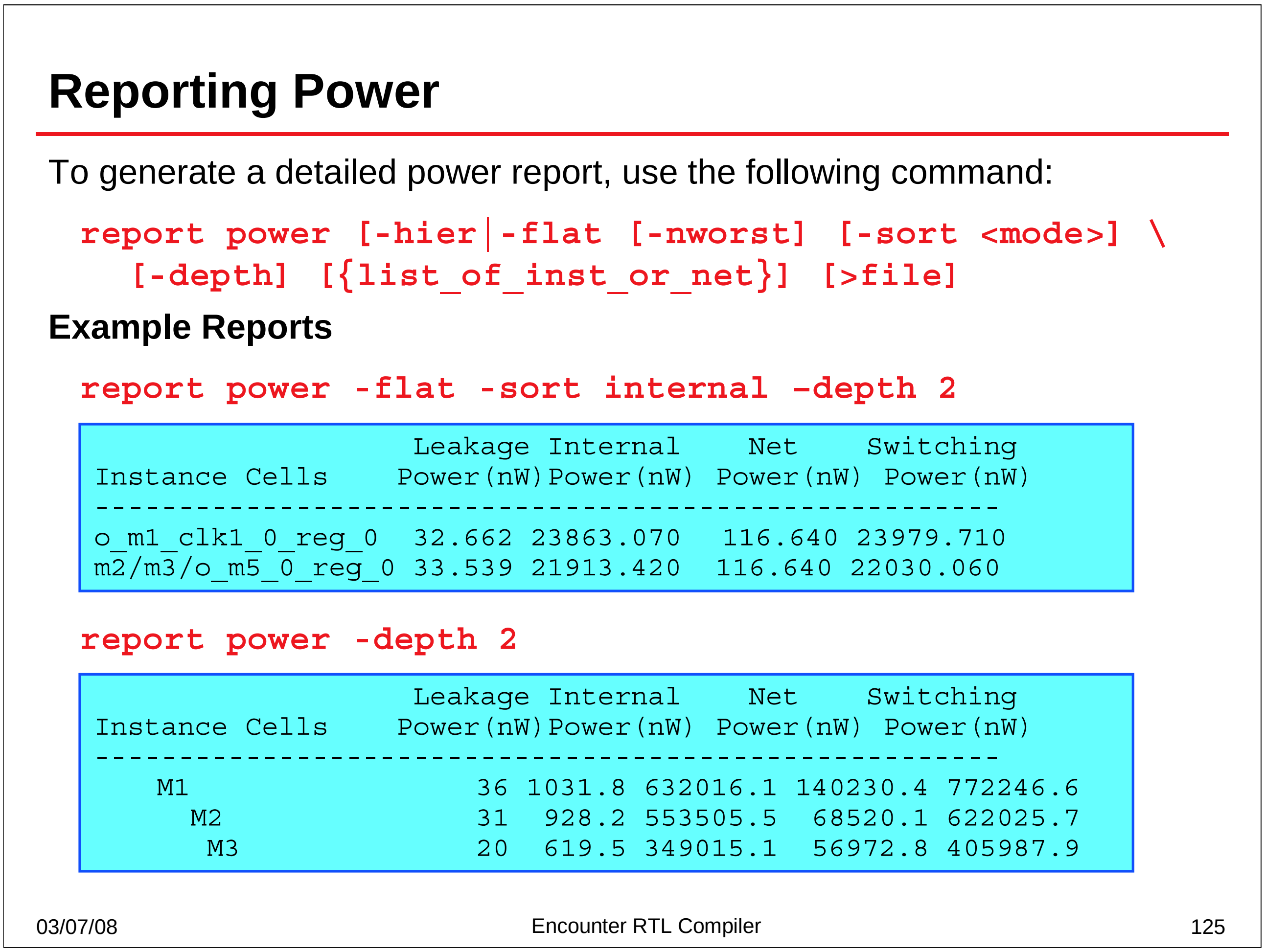
```
report clock_gating -ungated_ff
```

```
Ungated Flip-flops
---------------------------------
| Module | Flip-flop | Excluded |
---------------------------------
| alu    | zero_reg  | false |
---------------------------------
Total ungated flip-flops: 1
```

Clock Gating: Reporting a Clock-Gating Instance

```
report clock_gating -cg_instance RC_CG_HIER_INST_0
```

```
Clock Gating Instance : RC_CG_HIER_INST_0
Libcell: CGIC_LP (cg)
Style: latch_posedge
Module: alu_1 (alu)
Inputs:
ck_in = clock_187
enable = ena_188
Outputs:
ck_out = n_89
Gated FFs:
| Module  | Clock Gating Instance  | Fanout  | Gated Flip-flops |
|alu      | RC_CG_HIER_INST_0      |   8     | aluout_reg_0  |
|         |                        |         | aluout_reg_1  |
|         |                        |         | aluout_reg_2  |
|         |                        |         | aluout_reg_3  |
|         |                        |         | aluout_reg_4  |
|         |                        |         | aluout_reg_5  |
|         |                        |         | aluout_reg_6  |
|         |                        |         | aluout_reg_7  |
```



Reporting Power

To generate a detailed power report, use the following command:

```
report power [-hier|-flat [-nworst] [-sort <mode>] \
  [-depth] [{list_of_inst_or_net}] [>file]
```

Example Reports

```
report power -flat -sort internal -depth 2
```

```
                        Leakage Internal    Net     Switching
Instance Cells         Power(nW)Power(nW) Power(nW) Power(nW)
------------------------------------------------------
o_m1_clk1_0_reg_0   32.662 23863.070   116.640 23979.710
m2/m3/o_m5_0_reg_0 33.539 21913.420  116.640 22030.060
```

```
report power -depth 2
```

```
                        Leakage Internal    Net     Switching
Instance Cells         Power(nW)Power(nW) Power(nW) Power(nW)
------------------------------------------------------
    M1                     36 1031.8 632016.1 140230.4 772246.6
      M2                   31   928.2 553505.5   68520.1 622025.7
       M3                  20   619.5 349015.1   56972.8 405987.9
```

The following sort modes are available for instance-based power reports:

- Internal: Sorts by descending internal power.
- Leakage: (default) Sorts by descending leakage power.
- Net: Sorts by descending net power.
- Switching: Sorts by descending total switching power, which is the sum of the internal and net power.

You can also request net-based power reports.

Example

```
report power [find /designs/m1 -max 2 -net en*]
```

Obtain Power-Related Information

- To obtain the internal (dynamic) power on the design, use the following command:

```
get_attr lp_internal_power <instance|design>
```

- To get the leakage power on the design, use the following command:

```
get_attr lp_leakage_power <instance|design>
```

- To get the net power on the design, use the following command:

```
get_attr lp_net_power <instance|design|net>
```

- To find out the default toggle rate specified on the design, use this command:

```
get_attr lp_default_toggle_rate <design>
```

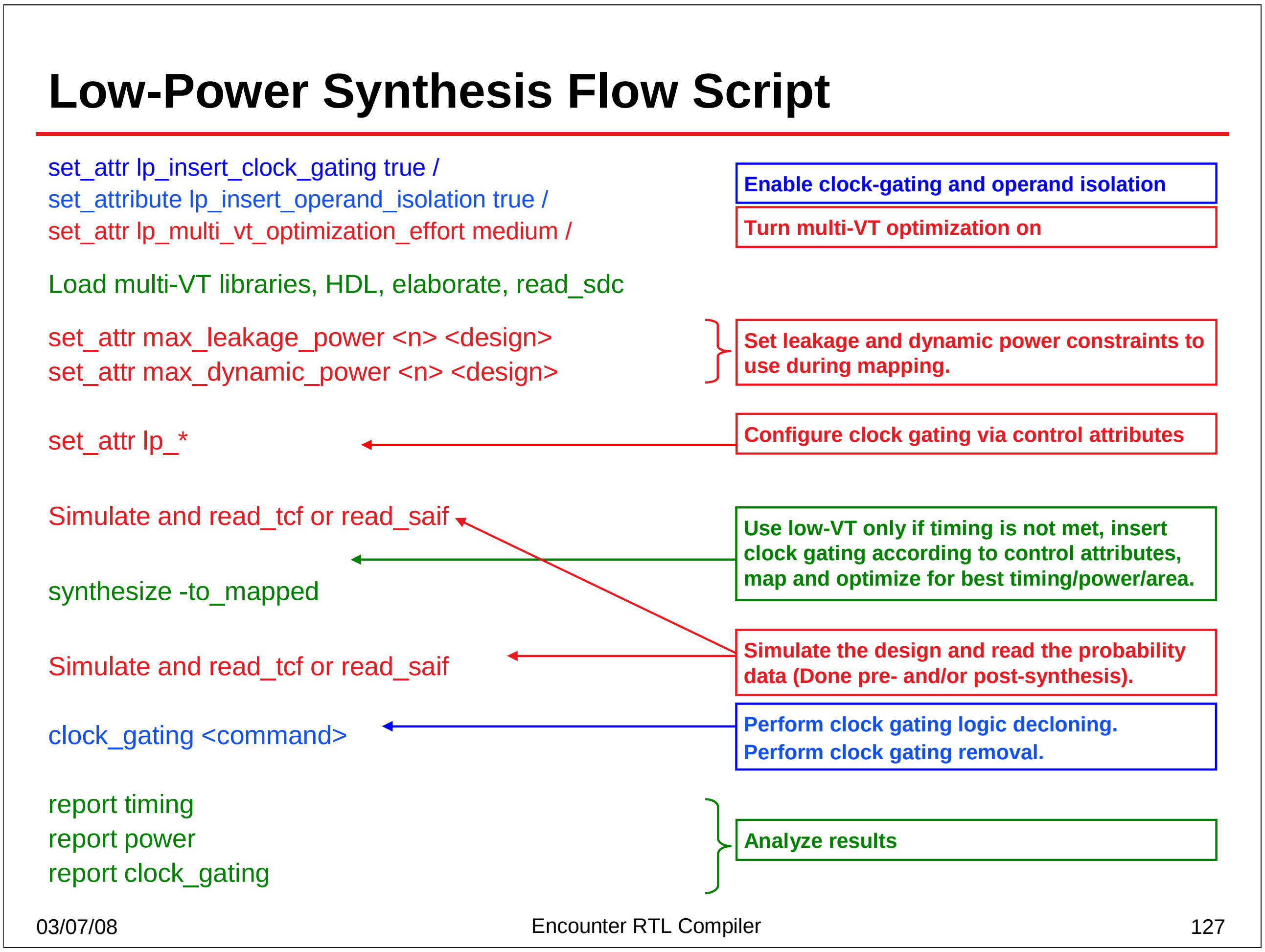
- To find out the default probability specified on the design, use the following command:

```
get_attr lp_default_probability <design>
```

To change the leakage power unit used for reporting, use the following command:

```
set_attribute lp_power_unit power_unit /
```

The power numbers shown in the log file are always given in *nW*. The value of the *lp_power_unit* root attribute does not affect the log file.



Low-Power Synthesis Flow Script

set_attr lp_insert_clock_gating true /
set_attribute lp_insert_operand_isolation true /
set_attr lp_multi_vt_optimization_effort medium /

Load multi-VT libraries, HDL, elaborate, read_sdc

set_attr max_leakage_power <n> <design>
set_attr max_dynamic_power <n> <design>

set_attr lp_*

Simulate and read_tcf or read_saif

synthesize -to_mapped

Simulate and read_tcf or read_saif

clock_gating <command>

report timing
report power
report clock_gating

Enable clock-gating and operand isolation

Turn multi-VT optimization on

Set leakage and dynamic power constraints to use during mapping.

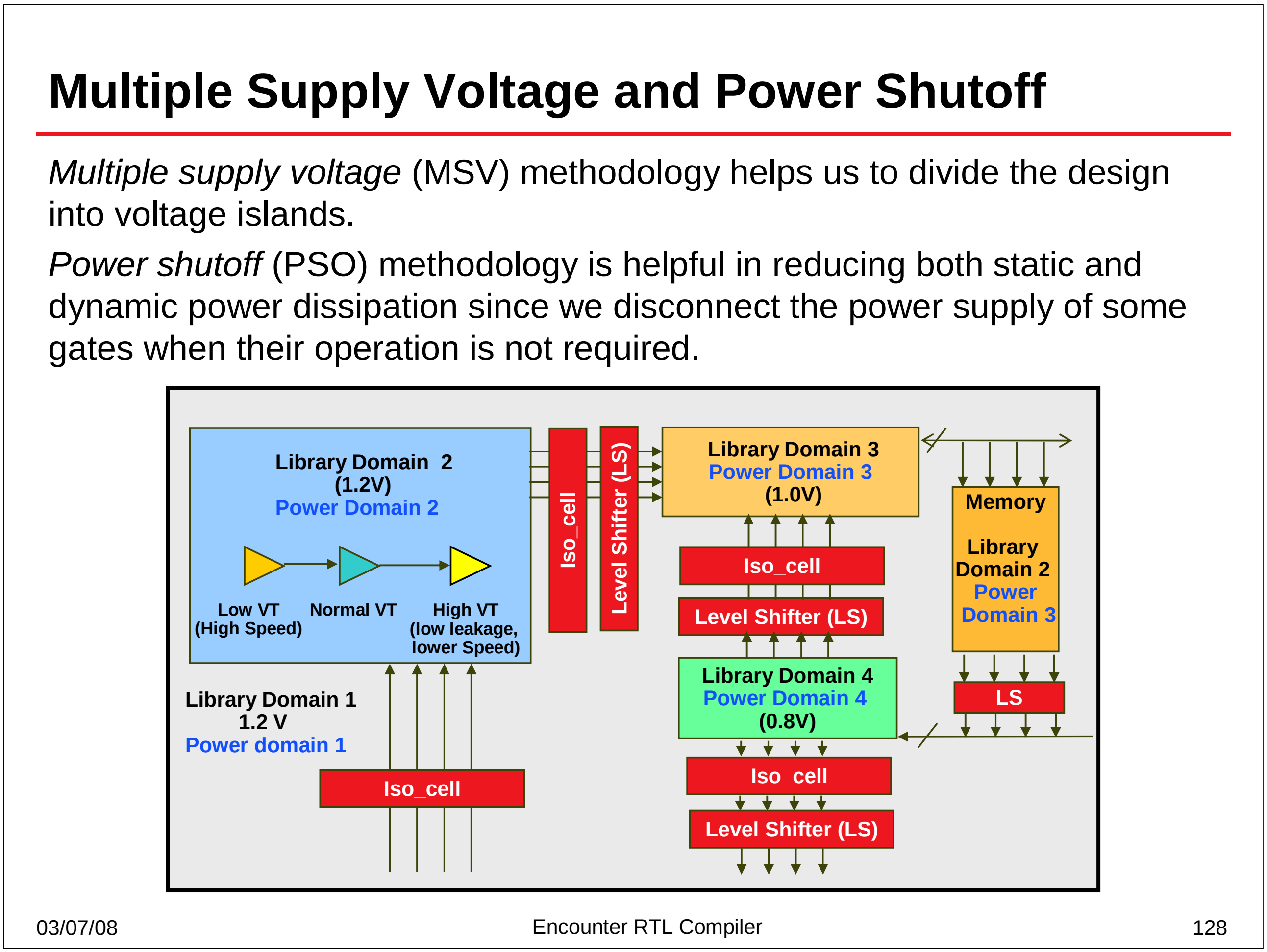
Configure clock gating via control attributes

Use low-VT only if timing is not met, insert clock gating according to control attributes, map and optimize for best timing/power/area.

Simulate the design and read the probability data (Done pre- and/or post-synthesis).

Perform clock gating logic decloning.
Perform clock gating removal.

Analyze results



Multiple Supply Voltage and Power Shutoff

Multiple supply voltage (MSV) methodology helps us to divide the design into voltage islands.

Power shutoff (PSO) methodology is helpful in reducing both static and dynamic power dissipation since we disconnect the power supply of some gates when their operation is not required.

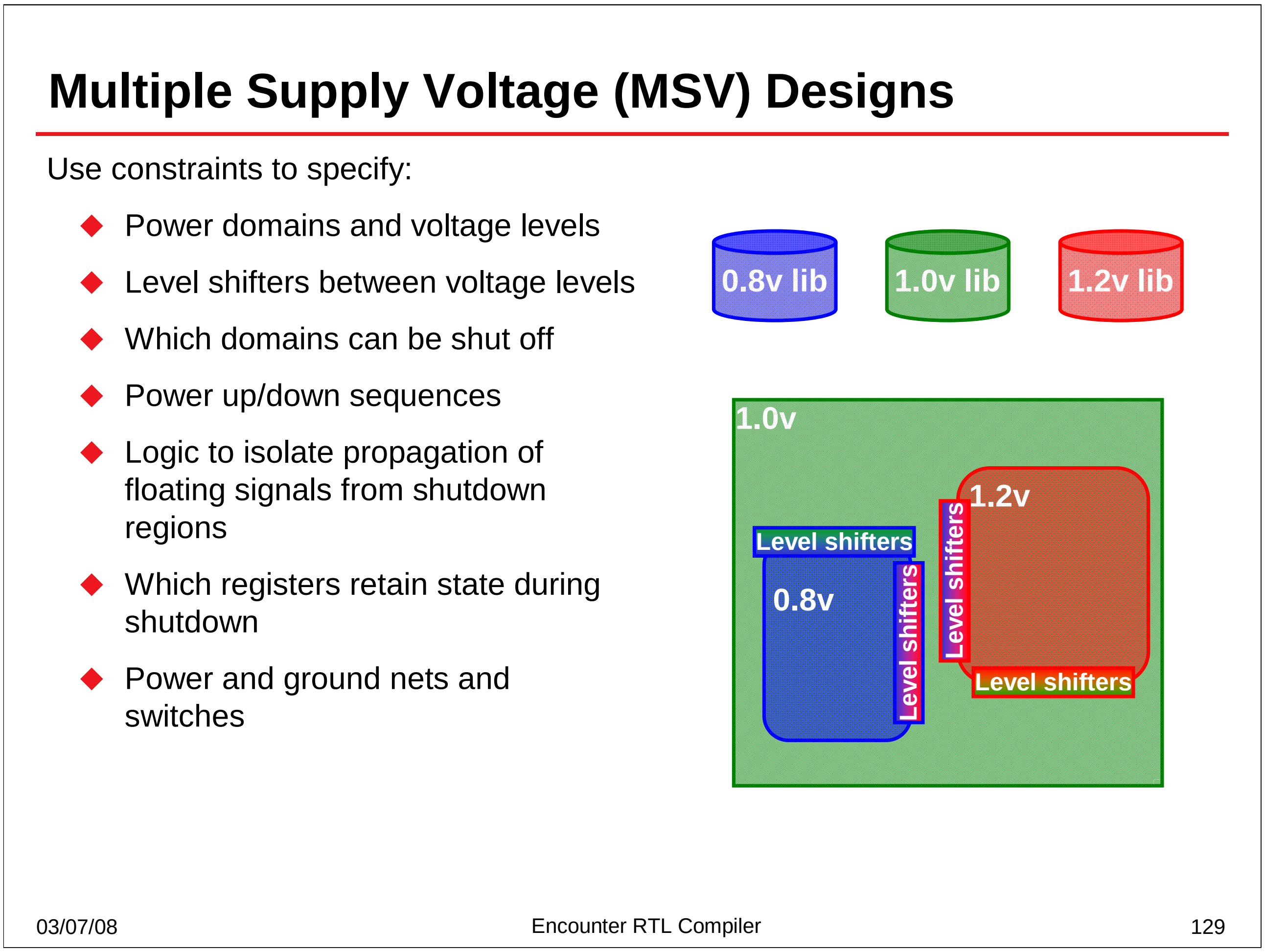
Reducing voltage level can greatly reduce power consumption.

Dynamic Power =$kC_LV^2f_{clk}$

When you use higher supply voltages, the design is faster, but the dynamic and leakage power increases. The MSV methodology assigns higher supply voltages to time-critical parts of the design and lower supply voltages to noncritical parts.

You can leverage power shutoff to:

- Shut off all switching activity
- Eliminate leakage power >10x



Multiple Supply Voltage (MSV) Designs

Use constraints to specify:

- Power domains and voltage levels
- Level shifters between voltage levels
- Which domains can be shut off
- Power up/down sequences
- Logic to isolate propagation of floating signals from shutdown regions
- Which registers retain state during shutdown
- Power and ground nets and switches

0.8v lib
1.0v lib
1.2v lib
1.0v
1.2v
Level shifters
Level shifters
0.8v
Level shifters
Level shifters

Reducing voltage levels reduces power.

- Reducing voltage also reduces speed.
- Requires architectural planning and implementation methodology support.

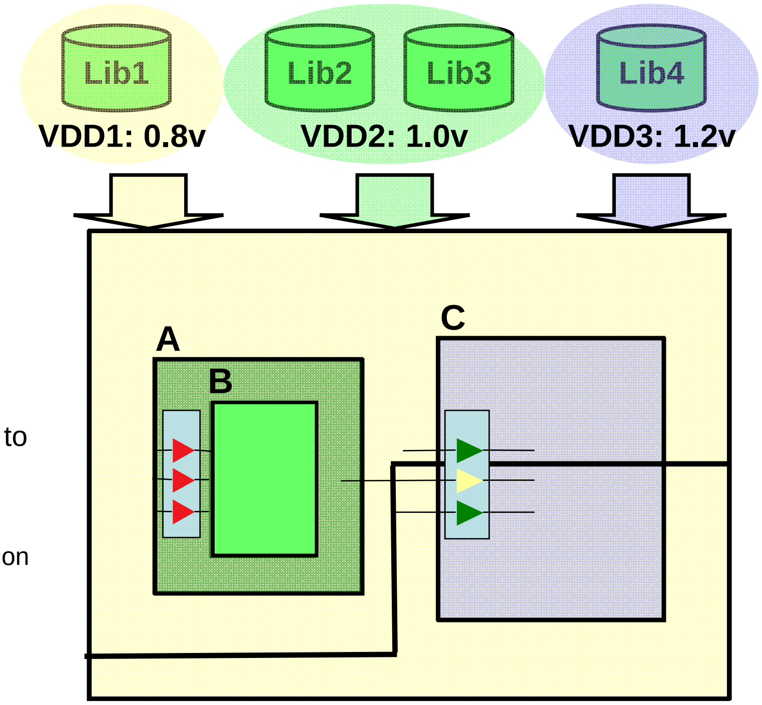
How do we:

- Determine the lowest voltage levels that can achieve frequency targets?
- Verify the implementation of voltage domains, level shifters, etc?
- Meet our time-to-market schedule while doing this?

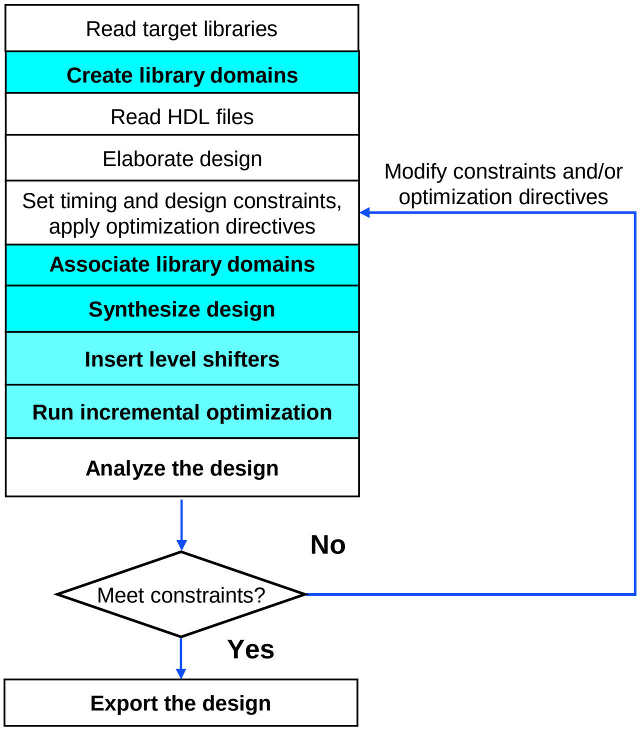
Top-down MSV Synthesis

MSV features include the following:

- Multiple voltage domains
  - Assign libraries to domains
  - Assign blocks to domains
- Top-down analysis and optimization
- Incremental what-if analysis
- Level shifter insertion
- Automatic transfer of voltage domain and shifter information to back-end tools
  - Place and Route
  - Low-power functional verification



MSV Synthesis Flow



Read target libraries

Create library domains

Read HDL files

Elaborate design

Set timing and design constraints, apply optimization directives

Associate library domains

Synthesize design

Insert level shifters

Run incremental optimization

Analyze the design

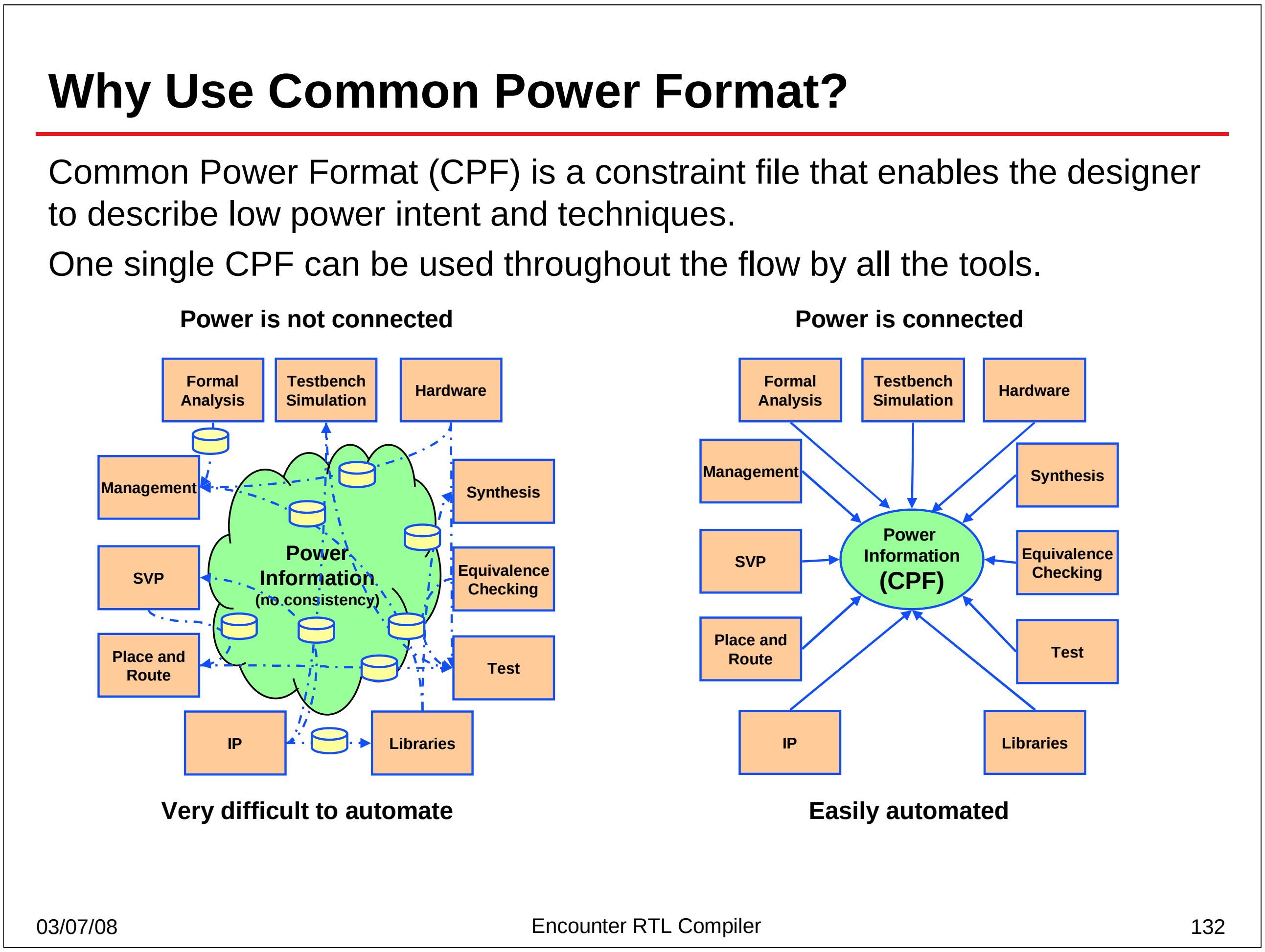
Meet constraints?

No

Modify constraints and/or optimization directives

Yes

Export the design



Why Use Common Power Format?

Common Power Format (CPF) is a constraint file that enables the designer to describe low power intent and techniques.

One single CPF can be used throughout the flow by all the tools.

Power is not connected

Very difficult to automate

Power is connected

Easily automated

03/07/08 Encounter RTL Compiler 132

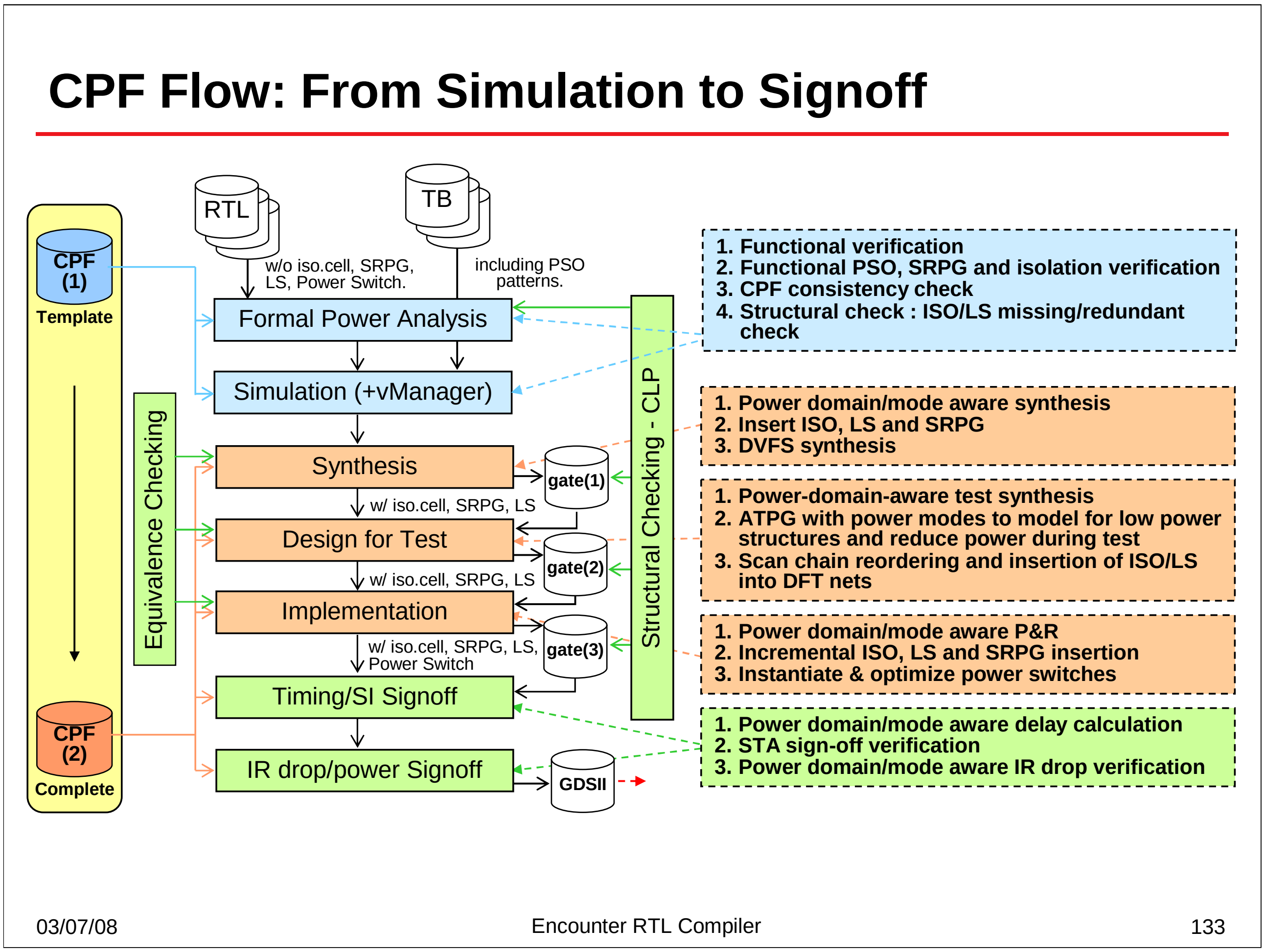
CPF Characteristics

- CPF is TCL-based.
- CPF reader contains both a linter and a parser.
- CPF language = TCL commands + CPF objects + Design objects

CPF Objects

- Power domain
- Analysis view
- Delay corner
- Library set
- Operating condition

Design objects already exist in the RTL/gate netlist: module, instance, net, pin, port, pad.



CPF Flow: From Simulation to Signoff

1. Functional verification
2. Functional PSO, SRPG and isolation verification
3. CPF consistency check
4. Structural check : ISO/LS missing/redundant check

1. Power domain/mode aware synthesis
2. Insert ISO, LS and SRPG
3. DVFS synthesis

1. Power-domain-aware test synthesis
2. ATPG with power modes to model for low power structures and reduce power during test
3. Scan chain reordering and insertion of ISO/LS into DFT nets

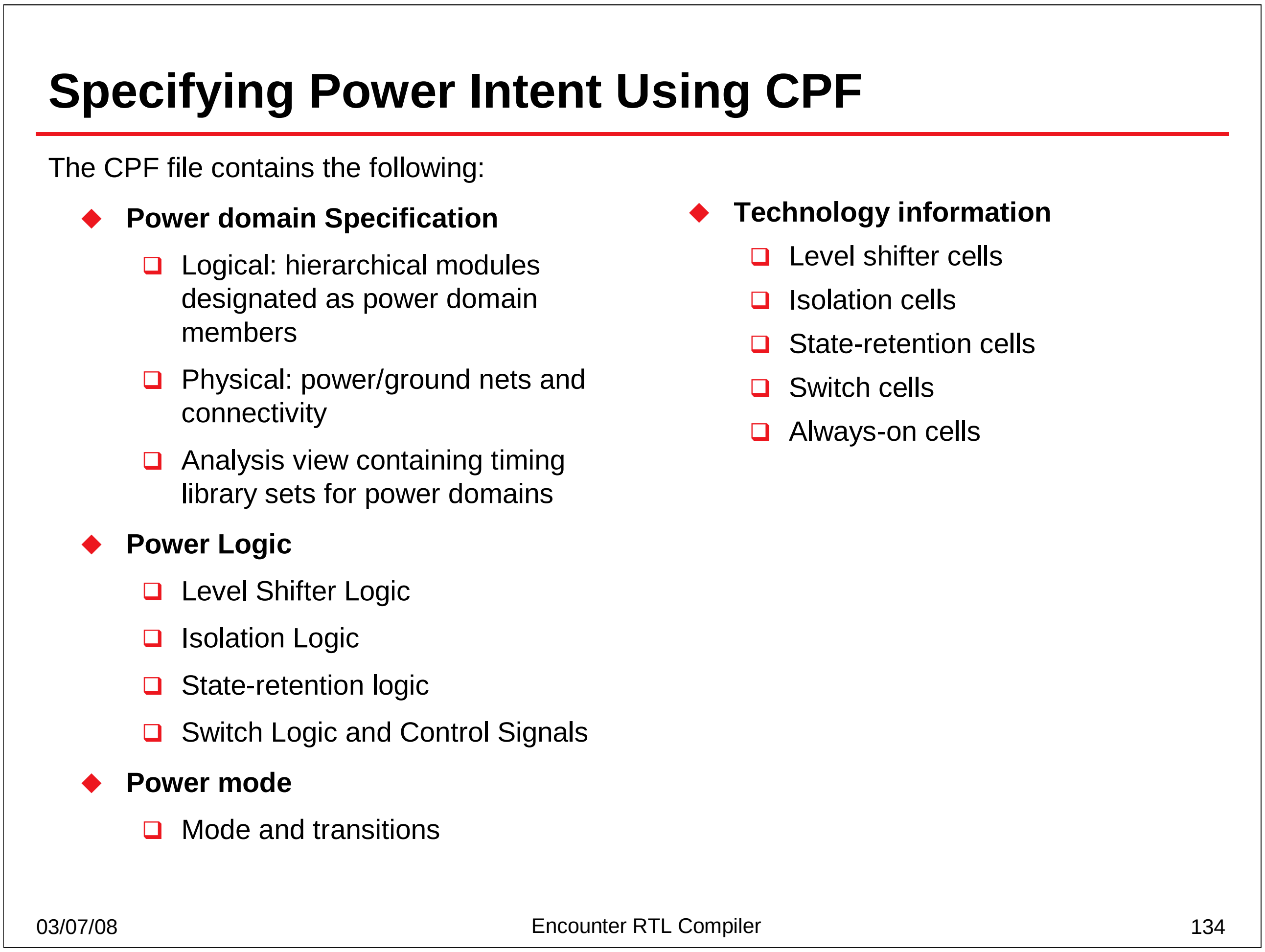
1. Power domain/mode aware P&R
2. Incremental ISO, LS and SRPG insertion
3. Instantiate & optimize power switches

1. Power domain/mode aware delay calculation
2. STA sign-off verification
3. Power domain/mode aware IR drop verification

03/07/08 Encounter RTL Compiler 133

DVFS: Dynamic Voltage Frequency Scaling

ATPG: Automatic Test Pattern Generation



Specifying Power Intent Using CPF

The CPF file contains the following:

- **Power domain Specification**
  - Logical: hierarchical modules designated as power domain members
  - Physical: power/ground nets and connectivity
  - Analysis view containing timing library sets for power domains
- **Power Logic**
  - Level Shifter Logic
  - Isolation Logic
  - State-retention logic
  - Switch Logic and Control Signals
- **Power mode**
  - Mode and transitions
- **Technology information**
  - Level shifter cells
  - Isolation cells
  - State-retention cells
  - Switch cells
  - Always-on cells

03/07/08 Encounter RTL Compiler 134

PSO (Power Shutoff)

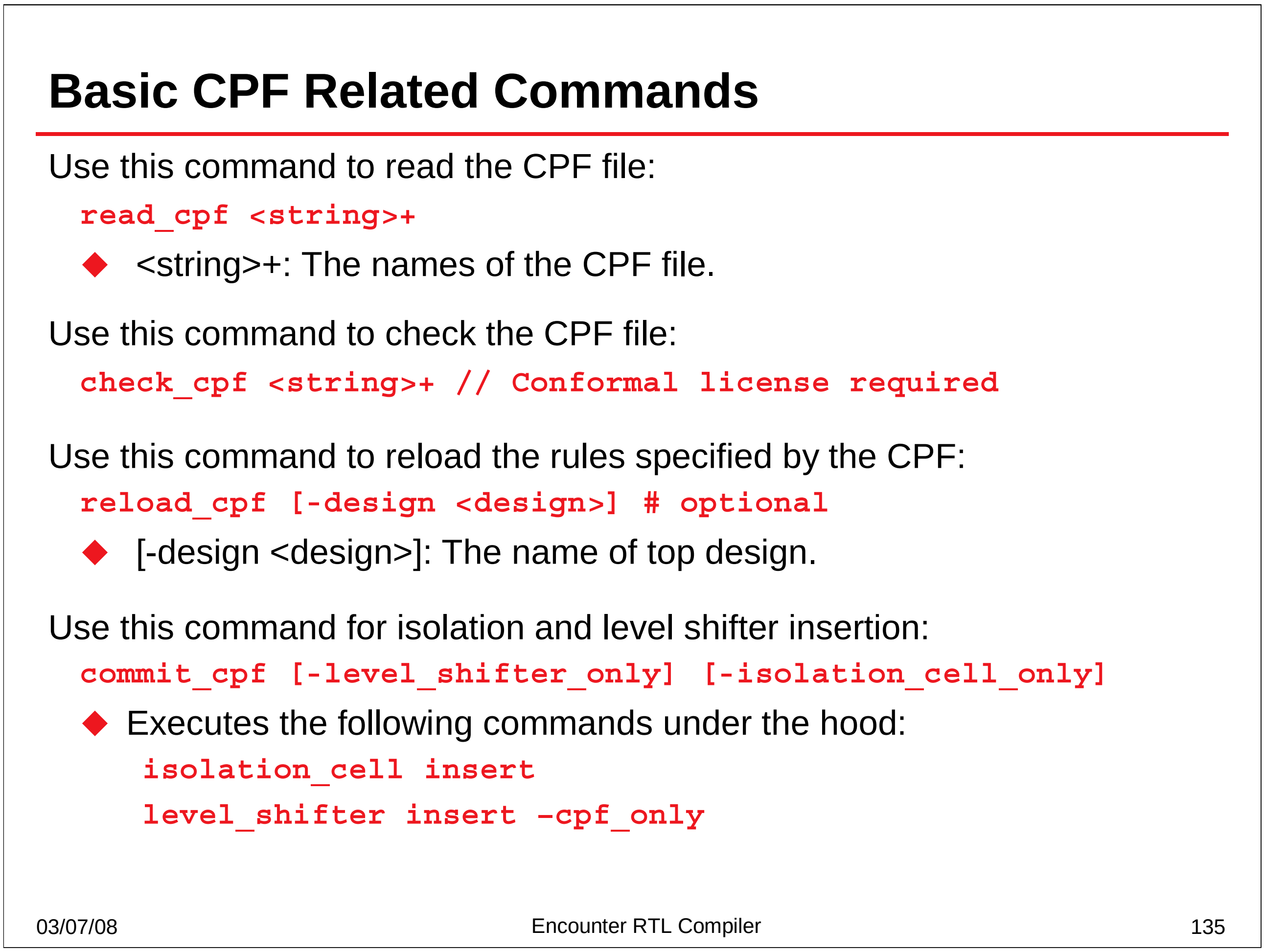
- Ability to switch off some blocks (power domains) in the design to save both dynamic and static power
- Need Isolation logic between on and off bocks (power domains) for correct functioning
- Need state retention logic in switched off blocks for saving the previous state

MSV (Multiple Supply Voltage)

- Ability to operate different blocks (power domains) of the design at different voltages to save dynamic and static power
- Need signal level shifters between these blocks for correct operation

DVFS (Dynamic Voltage Frequency Scaling)

- Ability to operate a block at more than one supply voltage
- Analysis of trade-off between power and timing. A block can operate at a lower supply voltage and dissipate less dynamic and static power when timing is not critical.



Basic CPF Related Commands

Use this command to read the CPF file:

```
read_cpf <string>+
```

- <string>+: The names of the CPF file.

Use this command to check the CPF file:

```
check_cpf <string>+ // Conformal license required
```

Use this command to reload the rules specified by the CPF:

```
reload_cpf [-design <design>] # optional
```

- [-design <design>]: The name of top design.

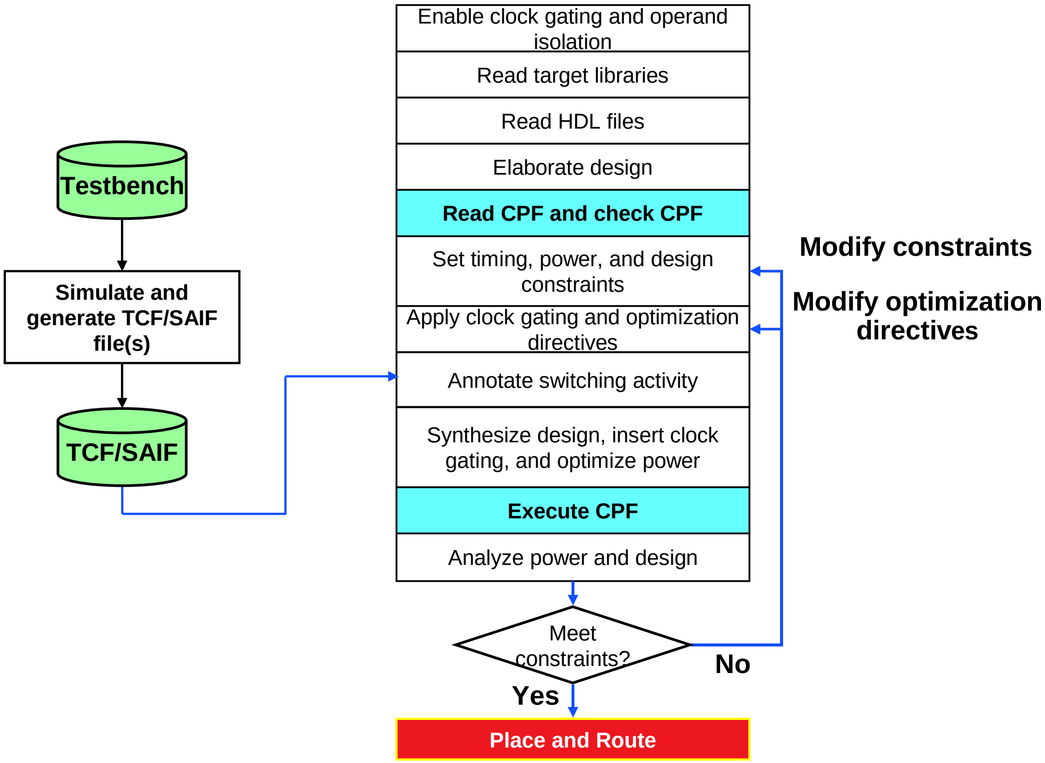
Use this command for isolation and level shifter insertion:

```
commit_cpf [-level_shifter_only] [-isolation_cell_only]
```

- Executes the following commands under the hood:

```
isolation_cell insert
level_shifter insert -cpf_only
```

Low-Power Flow Using CPF



Testbench

Simulate and generate TCF/SAIF file(s)

TCF/SAIF

Enable clock gating and operand isolation

Read target libraries

Read HDL files

Elaborate design

Read CPF and check CPF

Set timing, power, and design constraints

Apply clock gating and optimization directives

Annotate switching activity

Synthesize design, insert clock gating, and optimize power

Execute CPF

Analyze power and design

Meet constraints?

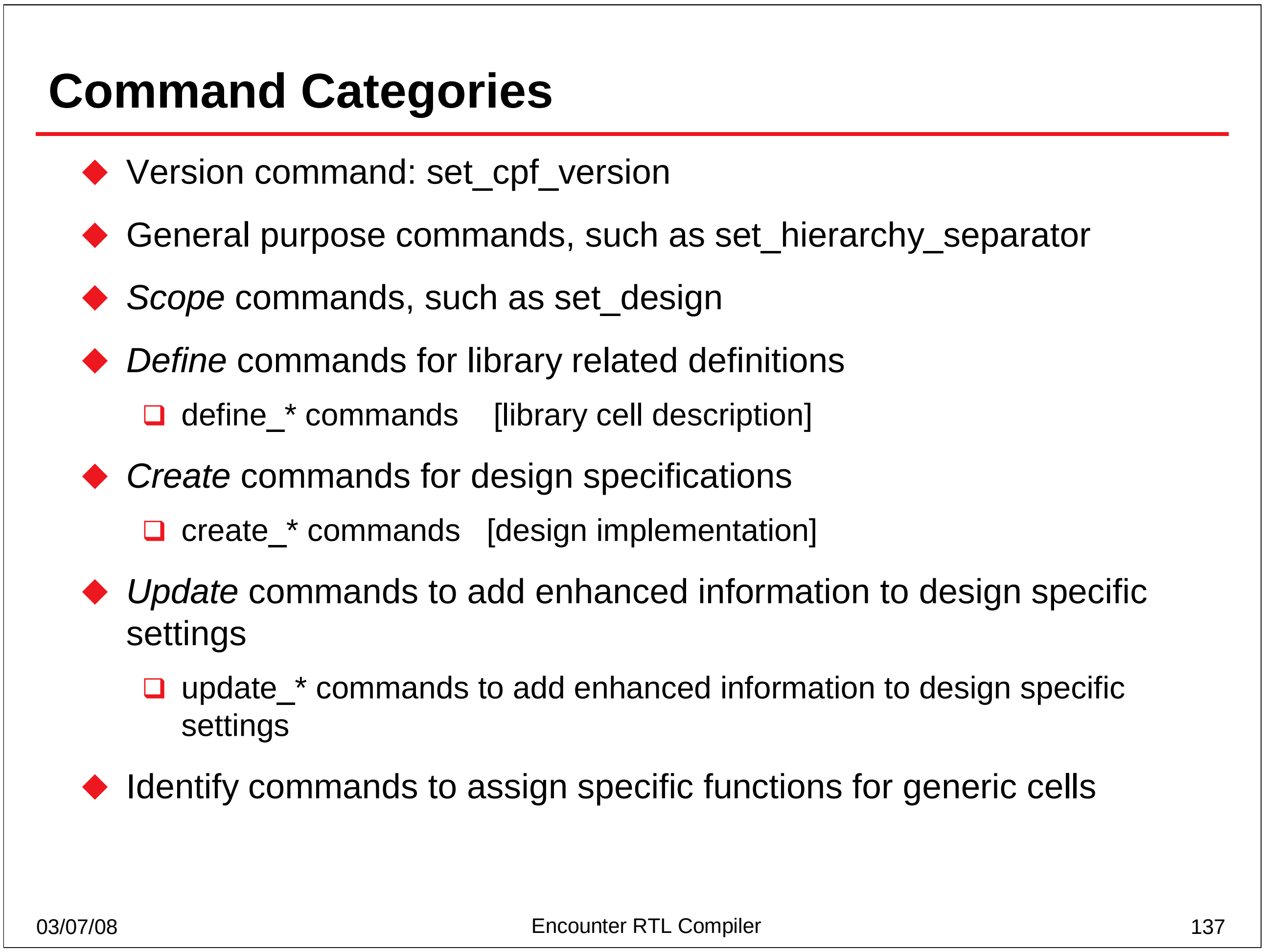
No

Modify constraints

Modify optimization directives

Yes

Place and Route



Command Categories

- Version command: set_cpf_version
- General purpose commands, such as set_hierarchy_separator
- *Scope* commands, such as set_design
- *Define* commands for library related definitions
  - define_* commands [library cell description]
- *Create* commands for design specifications
  - create_* commands [design implementation]
- *Update* commands to add enhanced information to design specific settings
  - update_* commands to add enhanced information to design specific settings
- Identify commands to assign specific functions for generic cells

- Create_* commands are typically used early in the architectural and simulation stages. They do not contain implementation details, such as net names, pin names.

```
create_power_domain -name domain -instances instance_list
```

- Update_* commands add the implementation details as they become available.

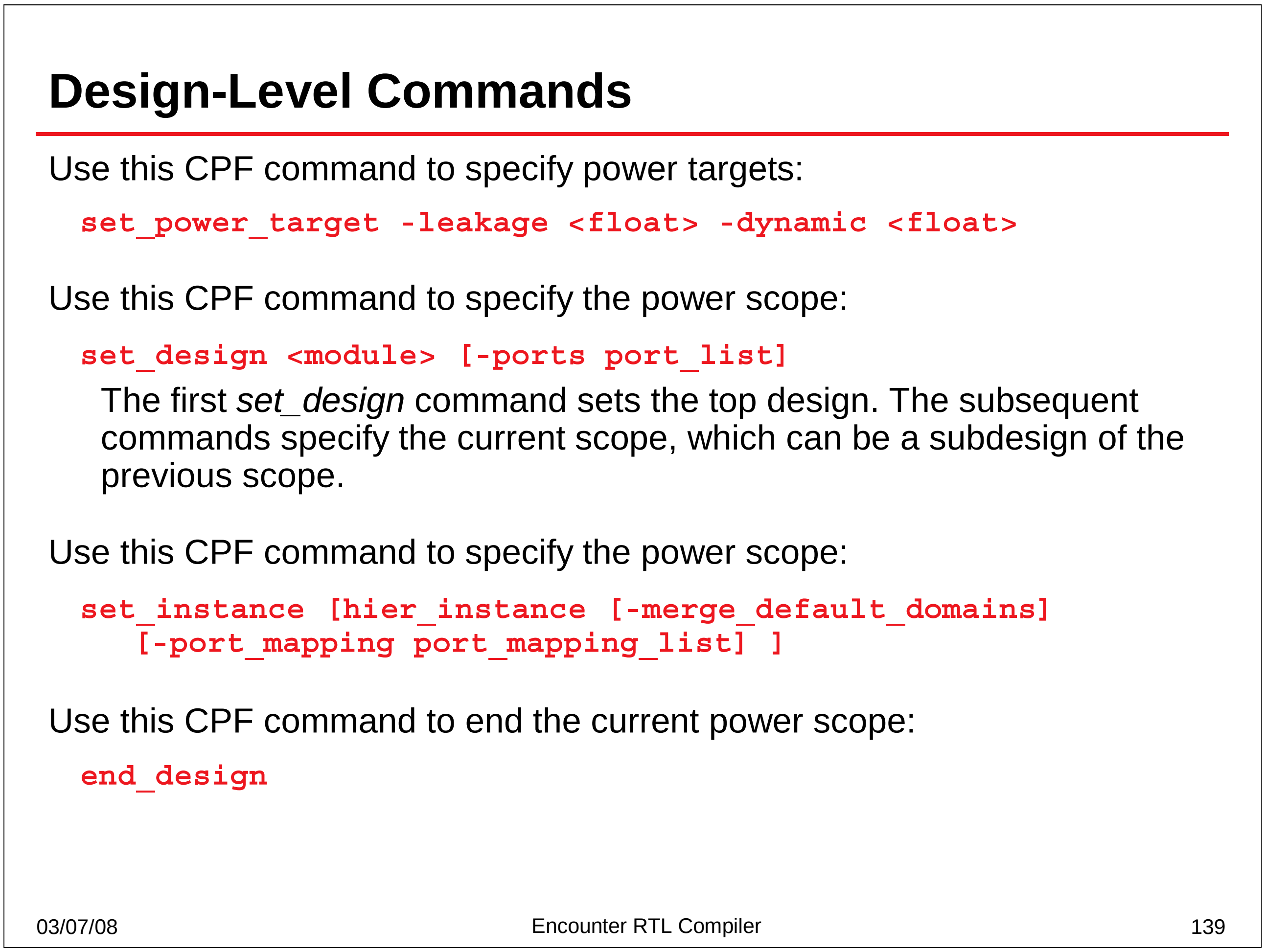
```
update_power_domain -name domain -internal_power_net net
```

Creating a Library Domain

If your design has multiple supply voltages, use the library domains to create multiple domains and assign them to different parts of the design for analysis and synthesis.

Use this CPF command to create a library set (library domain).

```
define_library_set -name library_domain -libraries
   <library_list>
```



Design-Level Commands

Use this CPF command to specify power targets:

```
set_power_target -leakage <float> -dynamic <float>
```

Use this CPF command to specify the power scope:

```
set_design <module> [-ports port_list]
```

The first *set_design* command sets the top design. The subsequent commands specify the current scope, which can be a subdesign of the previous scope.

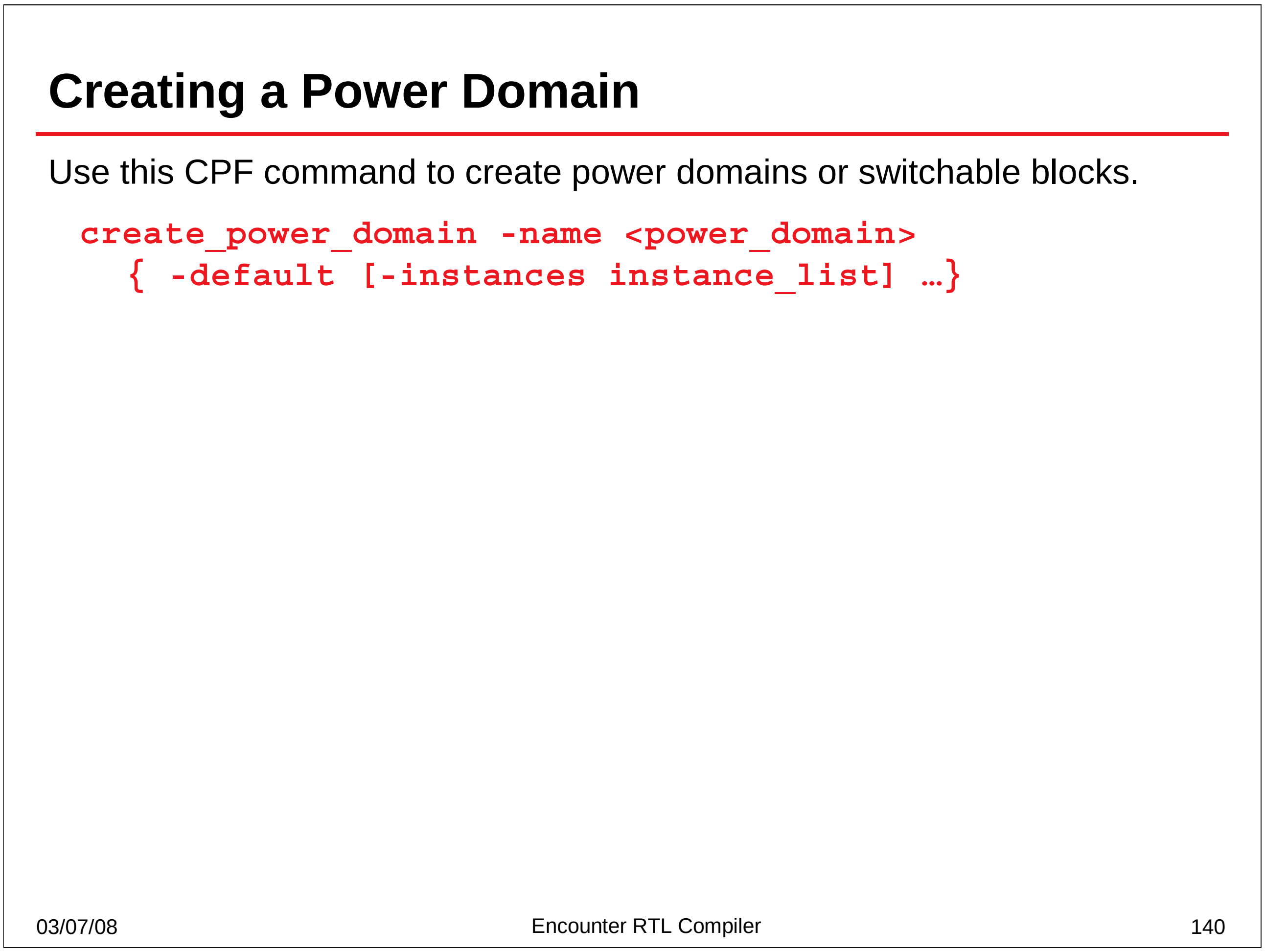
Use this CPF command to specify the power scope:

```
set_instance [hier_instance [-merge_default_domains]
   [-port_mapping port_mapping_list] ]
```

Use this CPF command to end the current power scope:

```
end_design
```

RTL Compiler connects the ports specified through the -ports option of the set_design command to the drivers specified through the -port_mapping option of the set_instance command. Therefore, the RC equivalent commands for 2 and 3 are not the exact match.



Creating a Power Domain

Use this CPF command to create power domains or switchable blocks.

```
create_power_domain -name <power_domain>
  { -default [-instances instance_list] …}
```

Library domains are logical, while power domains are physical.

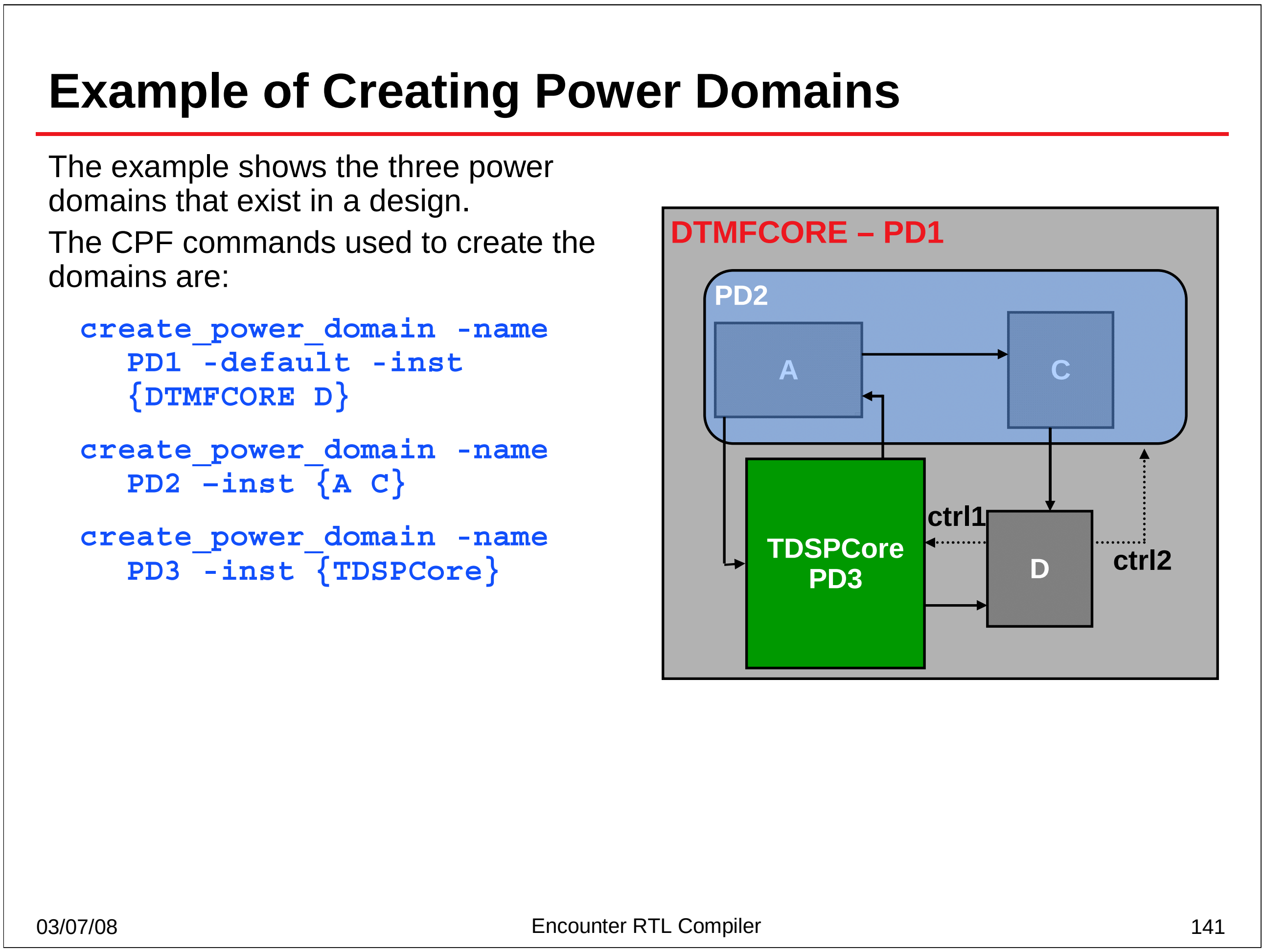
A power domain is a group of logic blocks that switch on and off at the same time and are connected to the same physical power supply line.

Power domains have the following constraints:

- Operate at only one supply voltage at any given time
- Are assigned only to a hierarchical instance, a pin, or a port
- Require isolation logic between them
- Obey a logical hierarchy and can be nested.

Power shutoff requires:

- **Shutoff signal**: To turn on or turn off the logic gates in the domain, such as SRPG cells or MTCMOS cells.
- **Isolation enable**: To enable or disable isolation cells.



Example of Creating Power Domains

The example shows the three power domains that exist in a design.

The CPF commands used to create the domains are:

```
create_power_domain -name
   PD1 -default -inst
   {DTMFCORE D}

create_power_domain -name
   PD2 -inst {A C}

create_power_domain -name
   PD3 -inst {TDSPCore}
```

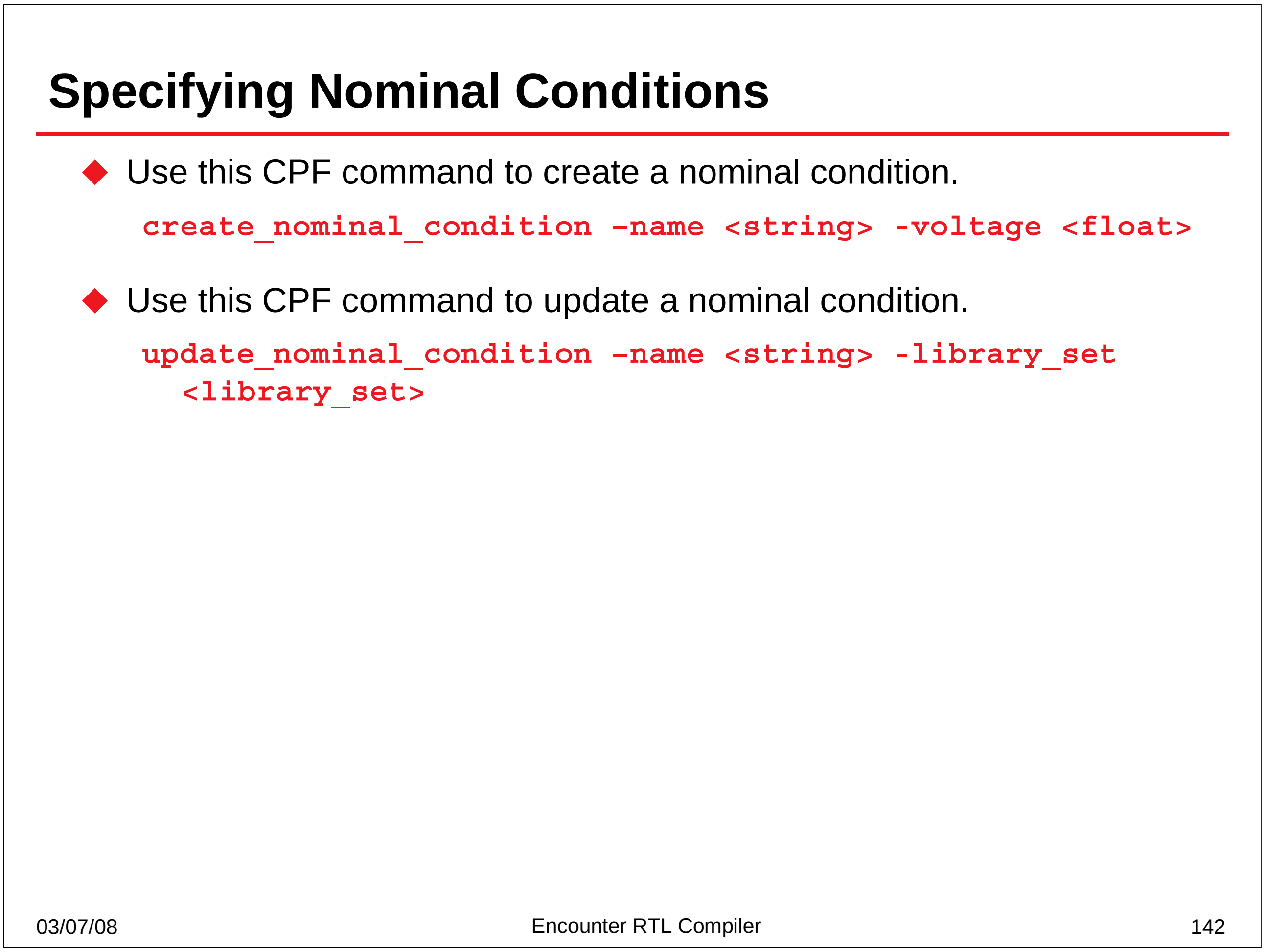
First, you need to define the power domains. In this example, you can see which blocks are assigned to which domains. The power domains need to follow the logical hierarchy, but can be nested. Each domain that you want to shut down needs a shutoff signal and a signal to isolate its outputs to active blocks when it is shut down.

You can update the power domain with the power and ground nets, using the following command:

```
update_power_domain -name <domain> -internal_power_net <net>
  -internal_ground_net <net>
```

RTL Compiler Equivalent

```
create_power_ground_nets -name <net> -power -internal
create_power_ground_nets -name <net> -ground -internal
```



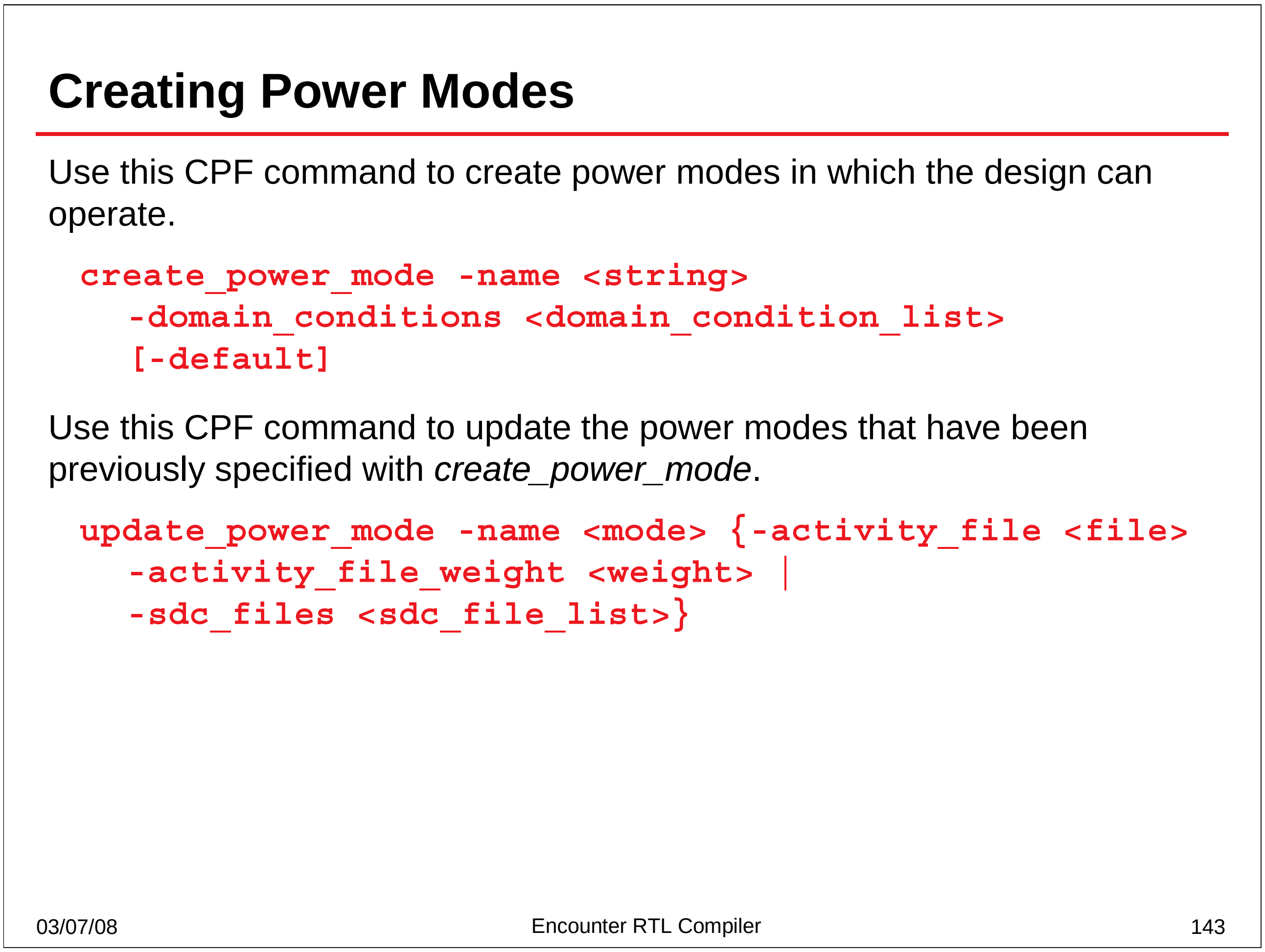
Specifying Nominal Conditions

- Use this CPF command to create a nominal condition.

```
create_nominal_condition -name <string> -voltage <float>
```

- Use this CPF command to update a nominal condition.

```
update_nominal_condition -name <string> -library_set
  <library_set>
```



Creating Power Modes

Use this CPF command to create power modes in which the design can operate.

```
create_power_mode -name <string>
   -domain_conditions <domain_condition_list>
   [-default]
```

Use this CPF command to update the power modes that have been previously specified with *create_power_mode*.

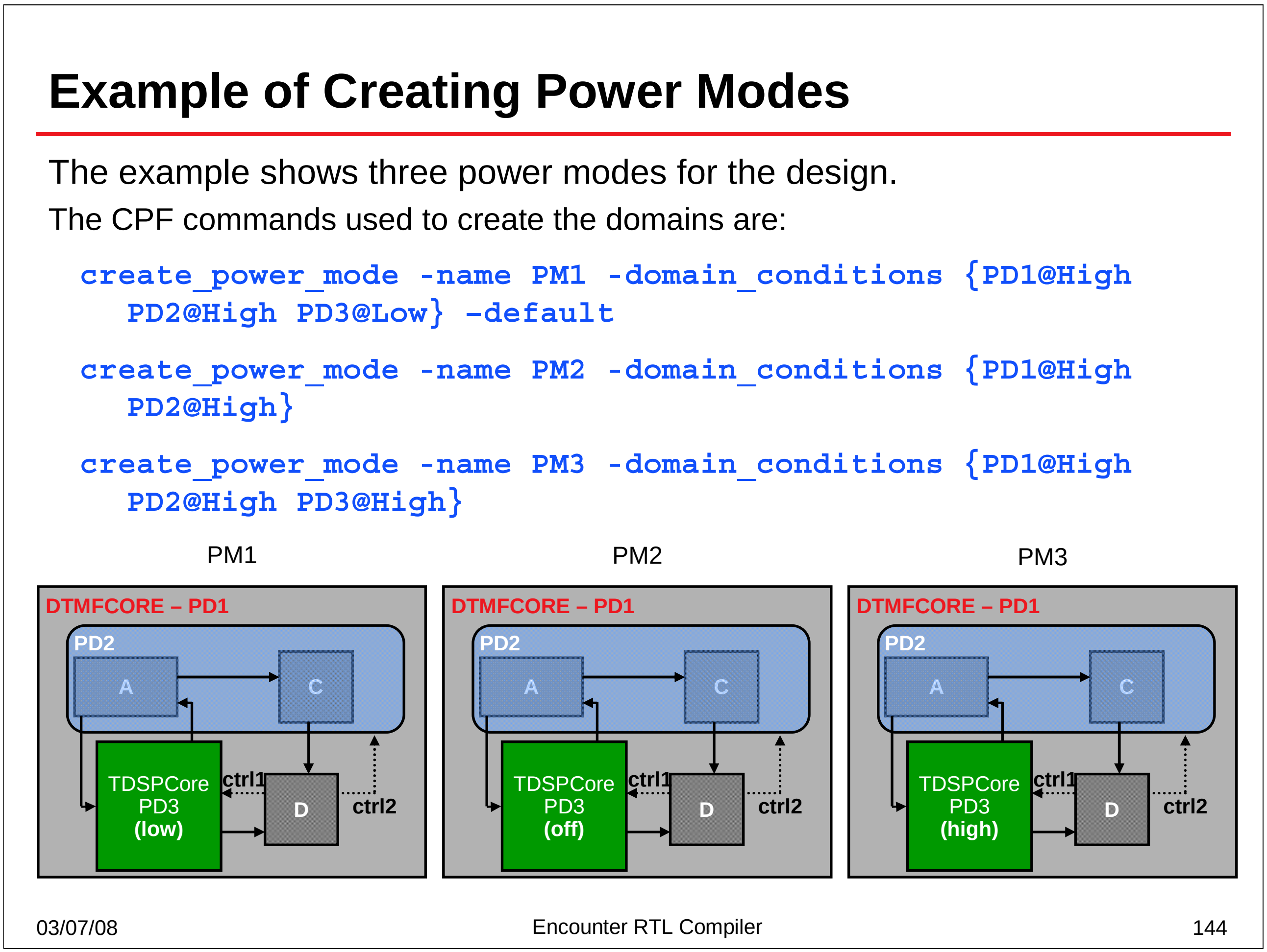
```
update_power_mode -name <mode> {-activity_file <file>
   -activity_file_weight <weight> |
   -sdc_files <sdc_file_list>}
```

03/07/08 Encounter RTL Compiler 143

For each nominal condition specified with the default mode, RTL Compiler looks at the corresponding *update_nominal_condition* command to find the associated library set for that condition, and links the corresponding library domain to the instances that belong to the power domain associated with this condition.

For a DVFS design (containing other modes that have different operating voltages compared to the voltages in default mode), RTL Compiler will set the following attribute for the other modes:

```
set_attribute library_domain_by_mode {{lib_domain mode}...}
  power_domain
```



Example of Creating Power Modes

The example shows three power modes for the design.

The CPF commands used to create the domains are:

```
create_power_mode -name PM1 -domain_conditions {PD1@High
   PD2@High PD3@Low} -default

create_power_mode -name PM2 -domain_conditions {PD1@High
   PD2@High}

create_power_mode -name PM3 -domain_conditions {PD1@High
   PD2@High PD3@High}
```

The first two commands show a regular MSV flow. The third command, when used in conjunction with first two commands, indicates that the design is being scaled for DVFS.

Specifying Power Modes

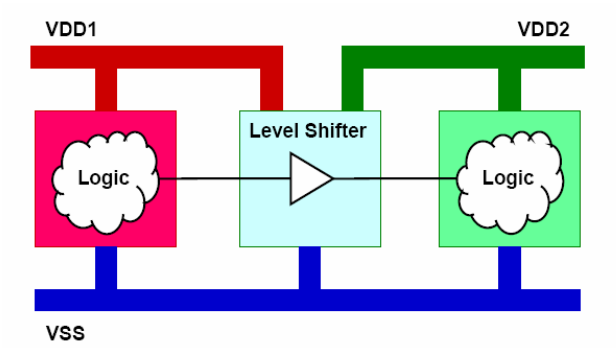
- Specify modes and mode transition.
- You can create a list of operation corners per mode.
- Power domain can have different voltage, libraries and constraints per mode.

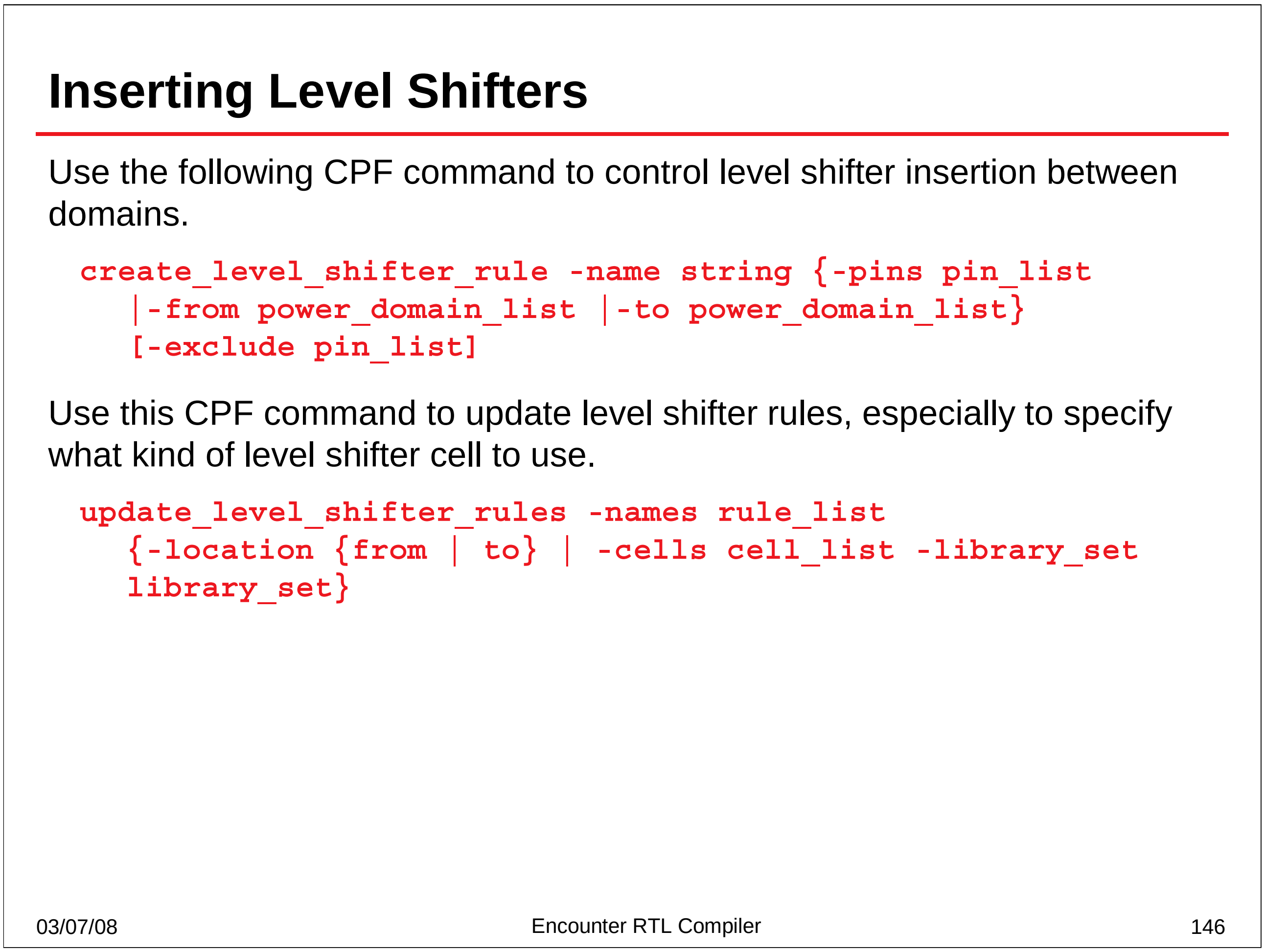
Specifying Level-Shifter Logic

Level shifters are required to shift signal levels from one voltage to another.

Use this CPF command to specify library cells to build level-shifting logic.

```
define_level_shifter_cell -cells <cell_list>
```





Inserting Level Shifters

Use the following CPF command to control level shifter insertion between domains.

```
create_level_shifter_rule -name string {-pins pin_list
   |-from power_domain_list |-to power_domain_list}
   [-exclude pin_list]
```

Use this CPF command to update level shifter rules, especially to specify what kind of level shifter cell to use.

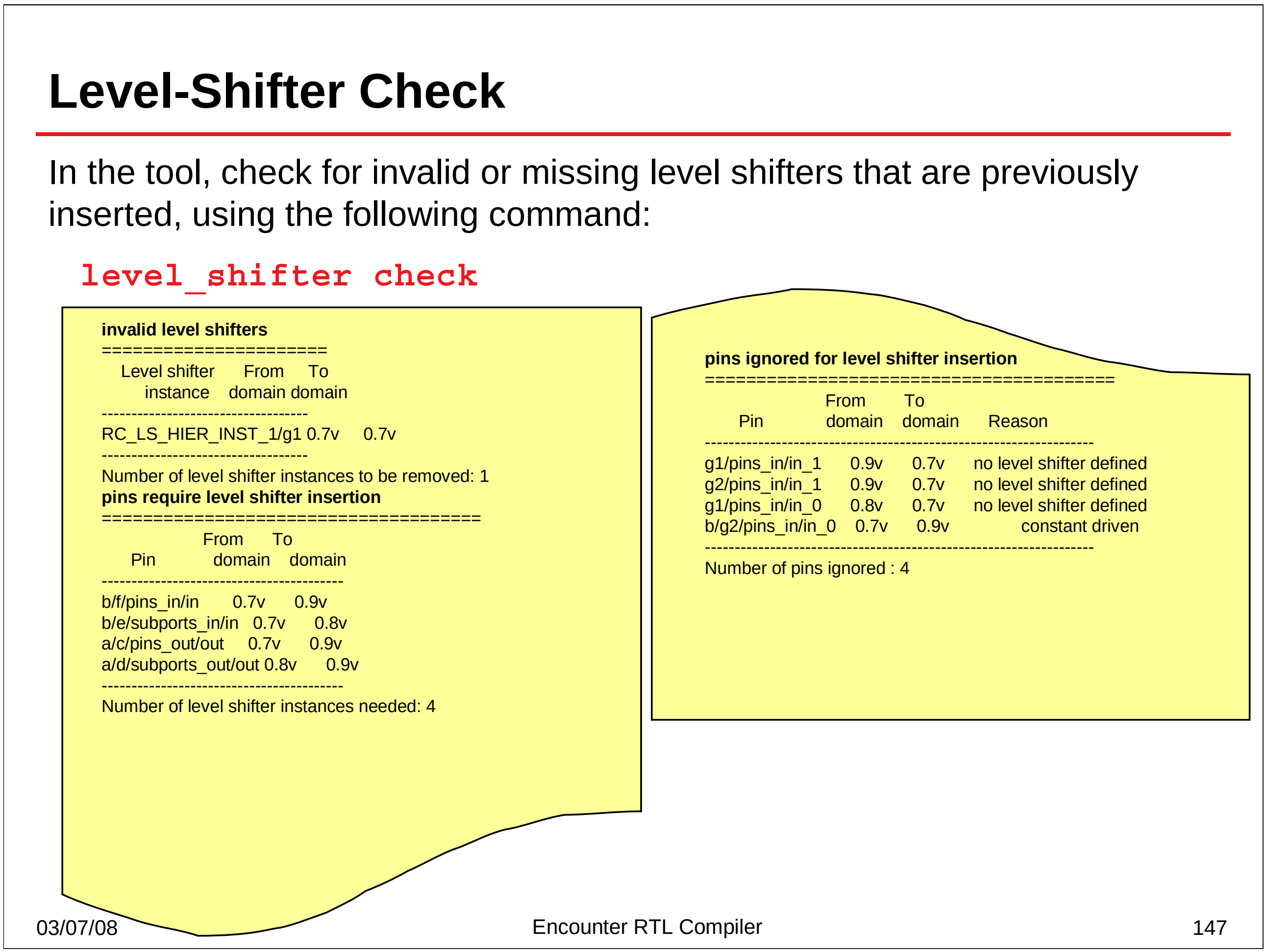
```
update_level_shifter_rules -names rule_list
   {-location {from | to} | -cells cell_list -library_set
   library_set}
```

Level Shifters

- Are inserted into a dedicated level of hierarchy.
- Are obtained in the destination library domain.
- Can be from low voltage to high voltage, or high to low.
- Can be inserted on ideal networks, including clock networks.

Level shifters cannot be inserted in these cases:

- At primary input/output
- On constant nets
- On floating nets/pins
- If there are multiple driver nets for location *to*, and if the drivers reside in different library domains
- If there are multiple fanout nets for location *from*, and if the fanouts reside in different library domains
- If no suitable level shifter cells are available



Level-Shifter Check

In the tool, check for invalid or missing level shifters that are previously inserted, using the following command:

```
level_shifter check
```

```
invalid level shifters
======================
   Level shifter      From     To
       instance    domain domain
-----------------------------------
RC_LS_HIER_INST_1/g1 0.7v     0.7v
-----------------------------------
Number of level shifter instances to be removed: 1
pins require level shifter insertion
===================================
                From      To
     Pin           domain    domain
-----------------------------------------
b/f/pins_in/in       0.7v      0.9v
b/e/subports_in/in   0.7v      0.8v
a/c/pins_out/out     0.7v      0.9v
a/d/subports_out/out 0.8v      0.9v
-----------------------------------------
Number of level shifter instances needed: 4
```

```
pins ignored for level shifter insertion
=======================================
                  From       To
      Pin         domain    domain      Reason
-----------------------------------------------------------------
g1/pins_in/in_1     0.9v      0.7v      no level shifter defined
g2/pins_in/in_1     0.9v      0.7v      no level shifter defined
g1/pins_in/in_0     0.8v      0.7v      no level shifter defined
b/g2/pins_in/in_0    0.7v      0.9v               constant driven
-----------------------------------------------------------------
Number of pins ignored : 4
```

To remove all the level shifters from the design, use:

```
level_shifter remove
```

To insert or update the level shifters in your design, or to remove the *unnecessary* level shifters from the design, use:

```
level_shifter update
```

Unnecessary implies the following:

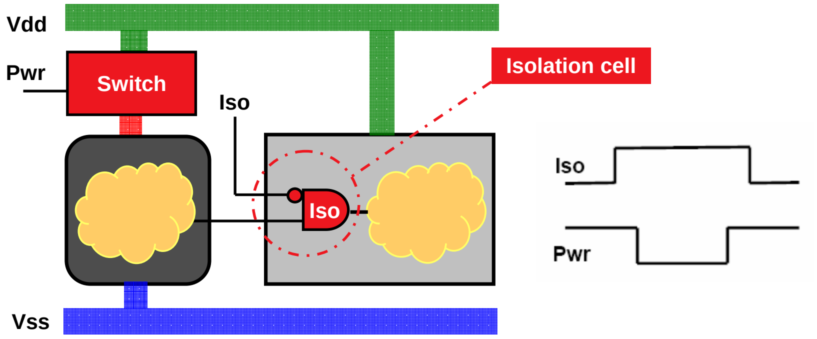
- The *driver* domain and *load* domain are the same.
- The library has more than one *driver* domain or more than one *load* domain.
- The level-shifter cell is driven by a constant.
- The level-shifter cell is not defined in level-shifter group.
- The level-shifter cell is a blackbox.
- The level-shifter cell is misused.

Specifying Isolation Logic

Isolation cells isolate the blocks that are powered off. Isolation precedes power off to prevent possible CMOS gate damage.

To specify library cells for isolation logic, use this CPF command:

```
define_isolation_cell -cells <cell_list>
```



Isolation cells isolate blocks that are powered off.

- Isolation precedes poweroff to prevent possible CMOS gate damage.
- Special cell to output a constant (1/0)
- Basic positioning rule: *From power-off domain to power-on domain*.

Defining Isolation Cell Rules

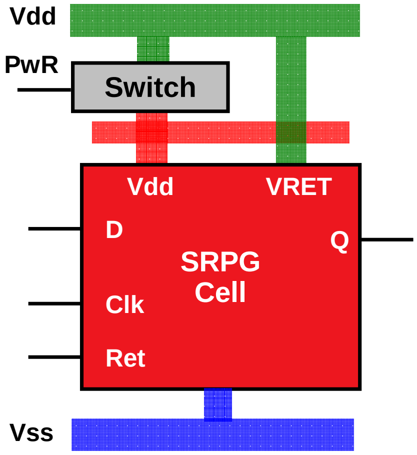
Use this CPF command to create isolation rules for isolation logic insertion.

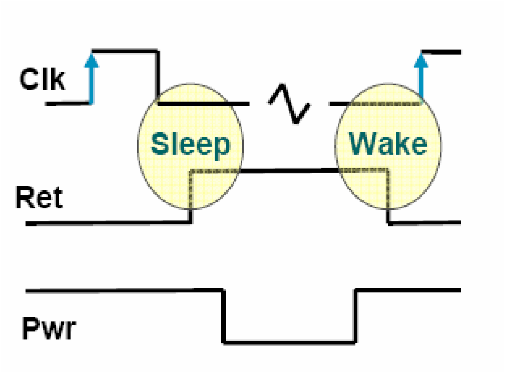
```
create_isolation_rule -name <string>
   -isolation_condition <expression>
   {-pins pin_list |-from power_domain_list
   |-to power_domain_list} [-isolation_target {from|to}]
   [-isolation_output {high|low|hold}]
   [-exclude pin_list]
```

State-Retention Power-Gated Register

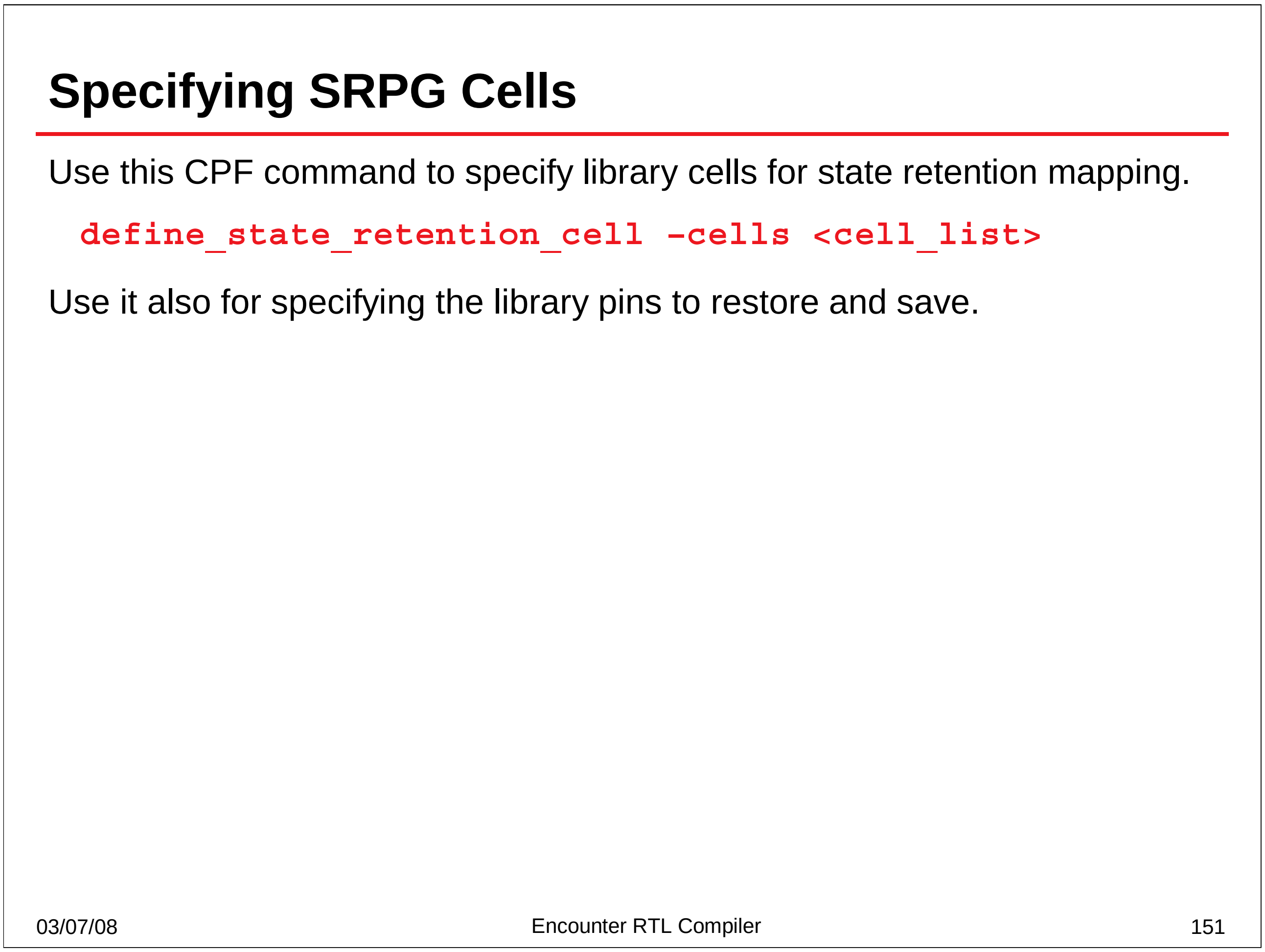
State-retention power-gated (SRPG) cells use retention voltage (VRET) to hold the current logic value when cells are turned off.

- Follow a power-down sequence for retention switching. First isolate, then save the state using the retention cells, and finally power down blocks.
- Modeled to output a 0 or 1 in retention mode.





State retention logic enables the system to recover the prior states after a power-up sequence.



Specifying SRPG Cells

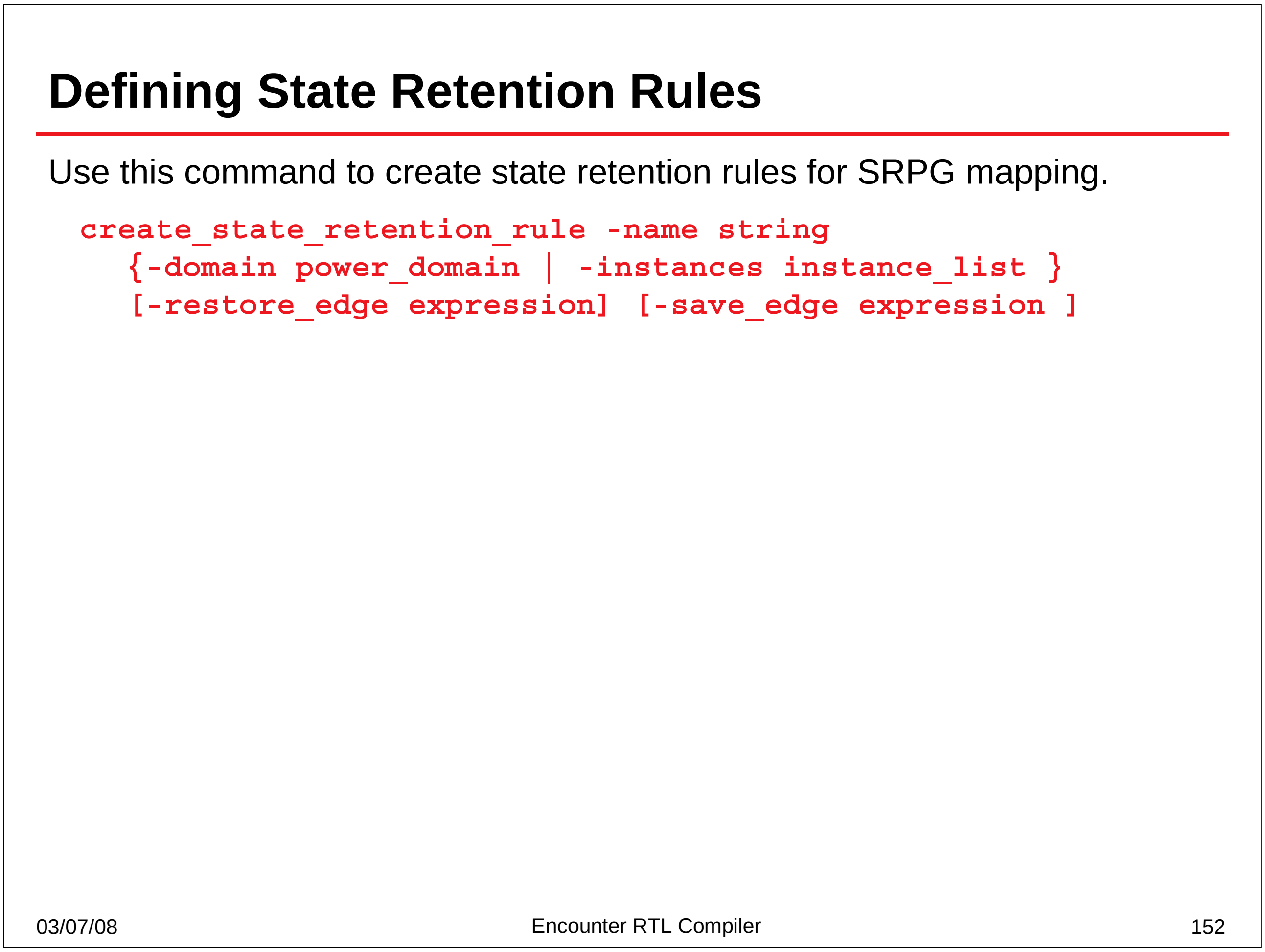
Use this CPF command to specify library cells for state retention mapping.

```
define_state_retention_cell -cells <cell_list>
```

Use it also for specifying the library pins to restore and save.

To specify a power switch, use:

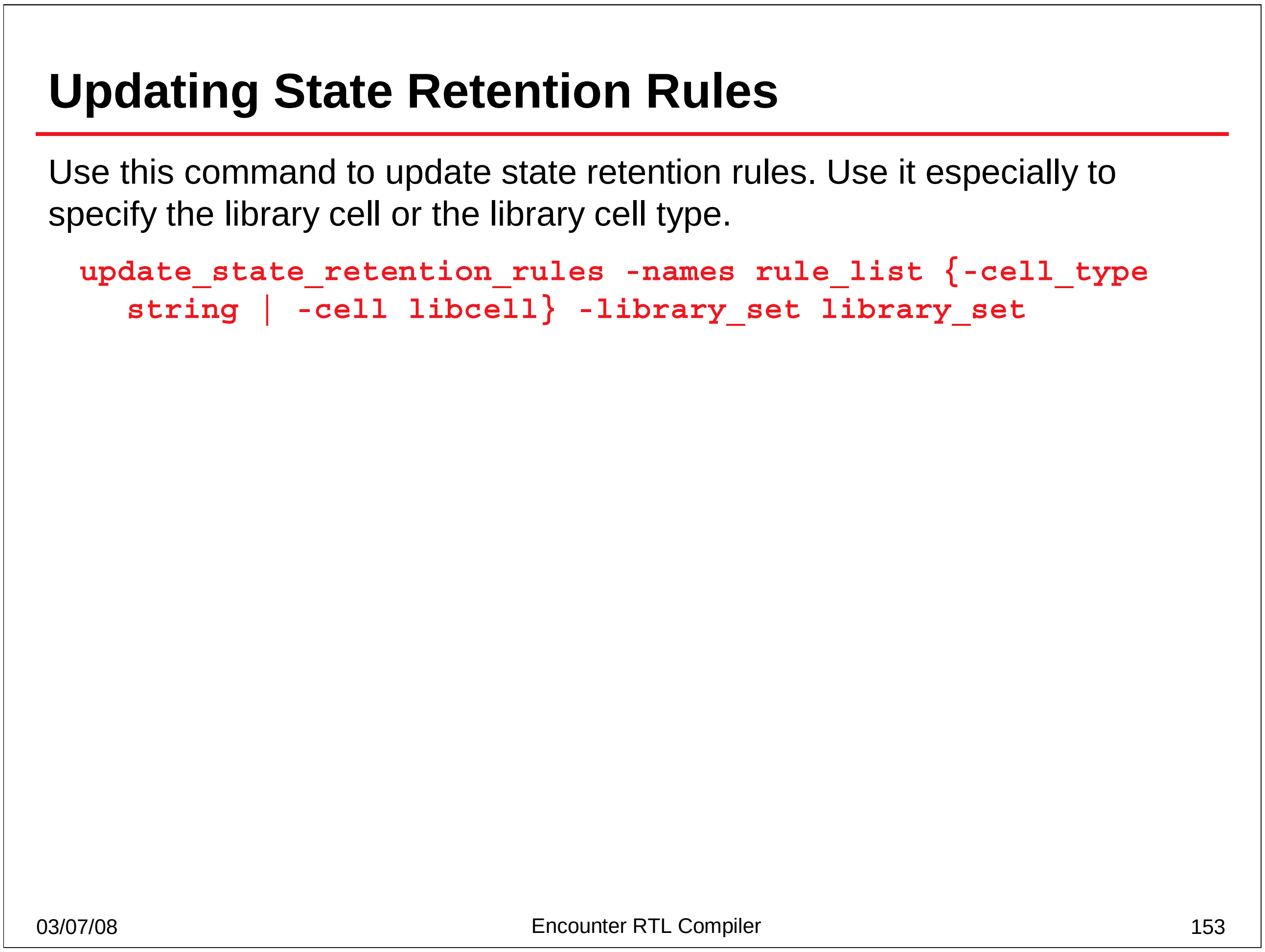
```
define_power_switch_cell -cells cell_list [-library_set library_set]
```



Defining State Retention Rules

Use this command to create state retention rules for SRPG mapping.

```
create_state_retention_rule -name string
   {-domain power_domain | -instances instance_list }
   [-restore_edge expression] [-save_edge expression ]
```

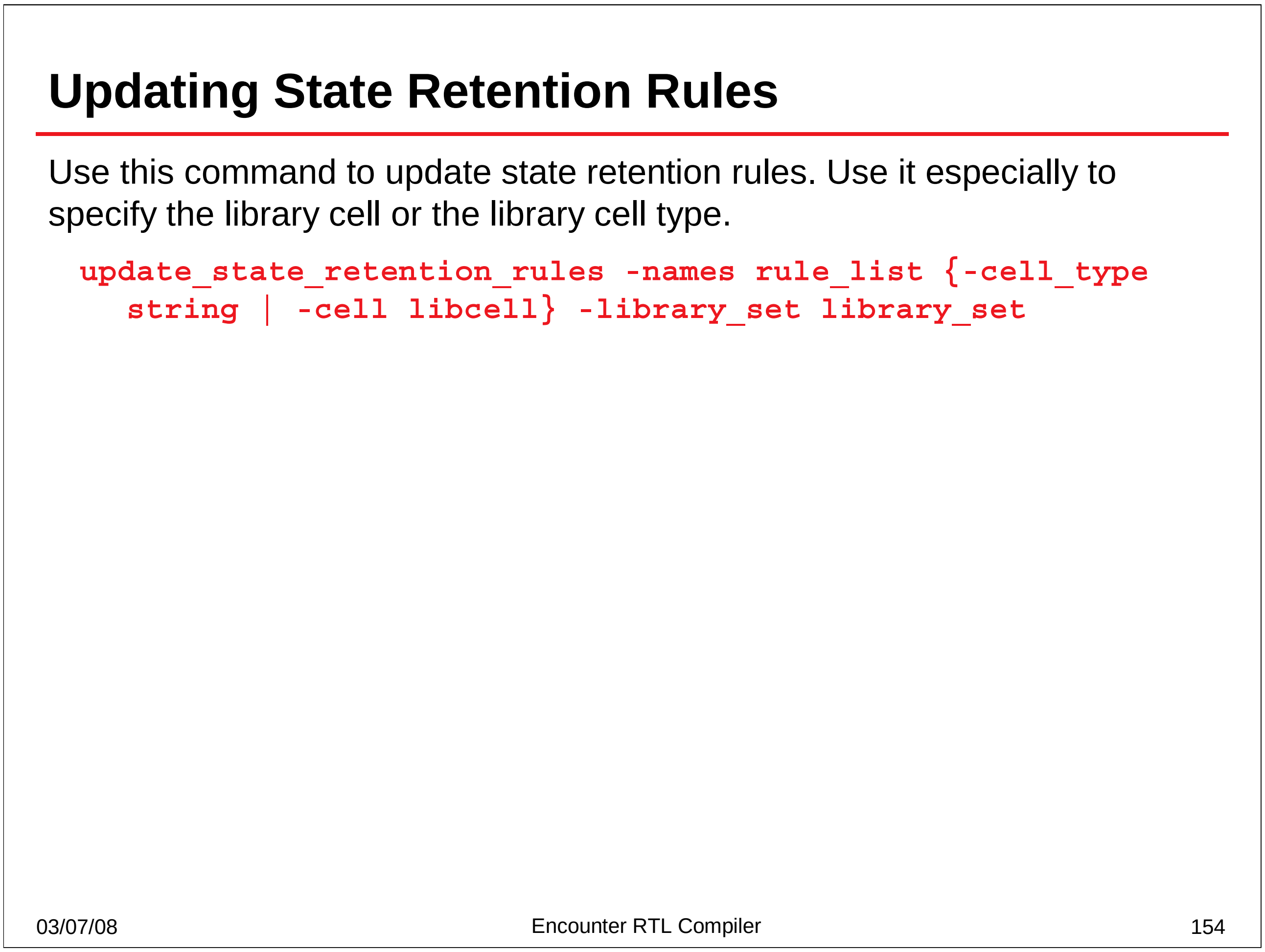


Updating State Retention Rules

Use this command to update state retention rules. Use it especially to specify the library cell or the library cell type.

```
update_state_retention_rules -names rule_list {-cell_type
  string | -cell libcell} -library_set library_set
```

Encounter$^{\hat{W}}$ RTL Compiler only maps to cells specified through the *define_state_retention_cell* command.

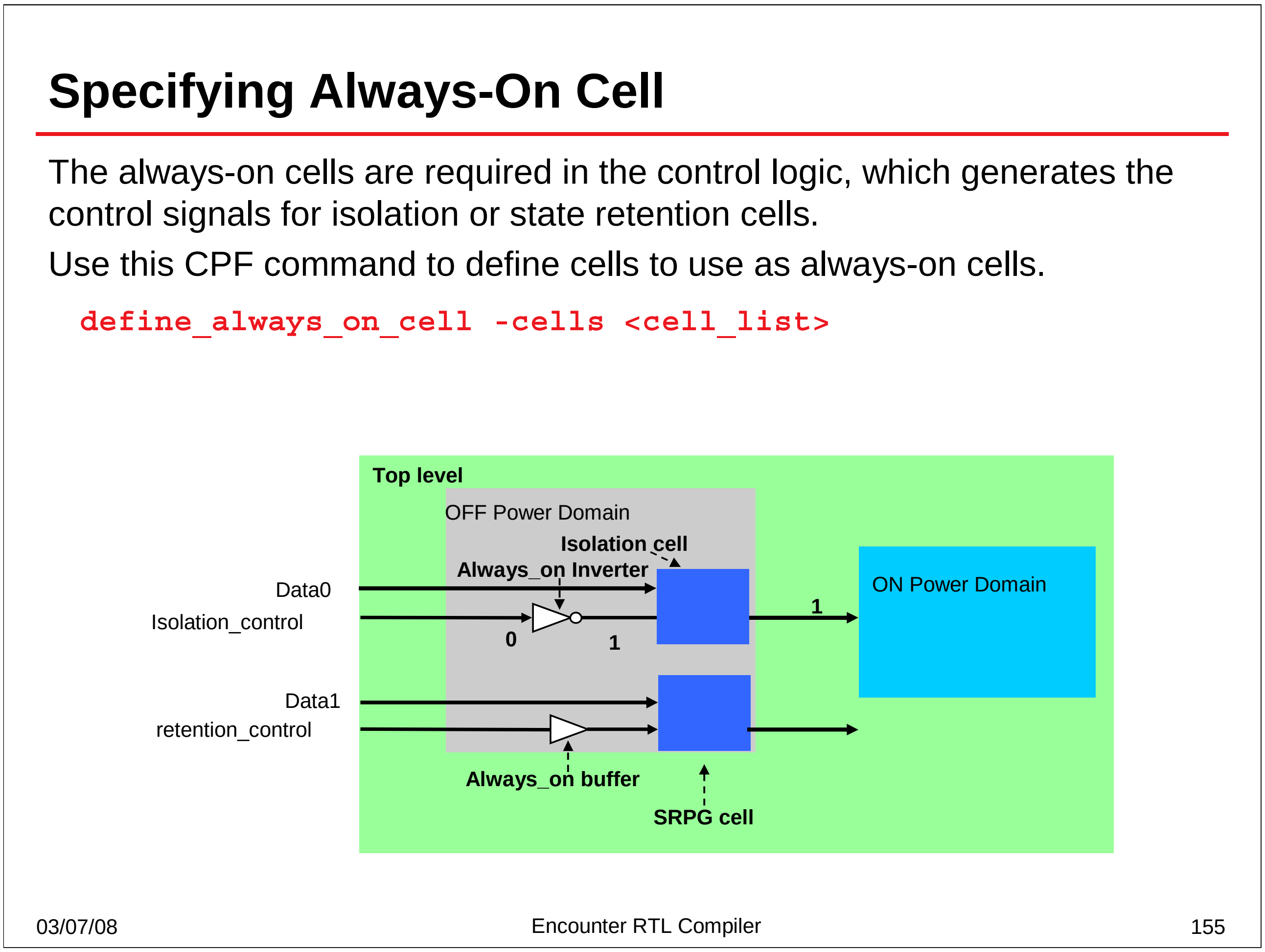


Updating State Retention Rules

Use this command to update state retention rules. Use it especially to specify the library cell or the library cell type.

```
update_state_retention_rules -names rule_list {-cell_type
   string | -cell libcell} -library_set library_set
```

Encounter® RTL Compiler only maps to cells specified through the *define_state_retention_cell* command.



Specifying Always-On Cell

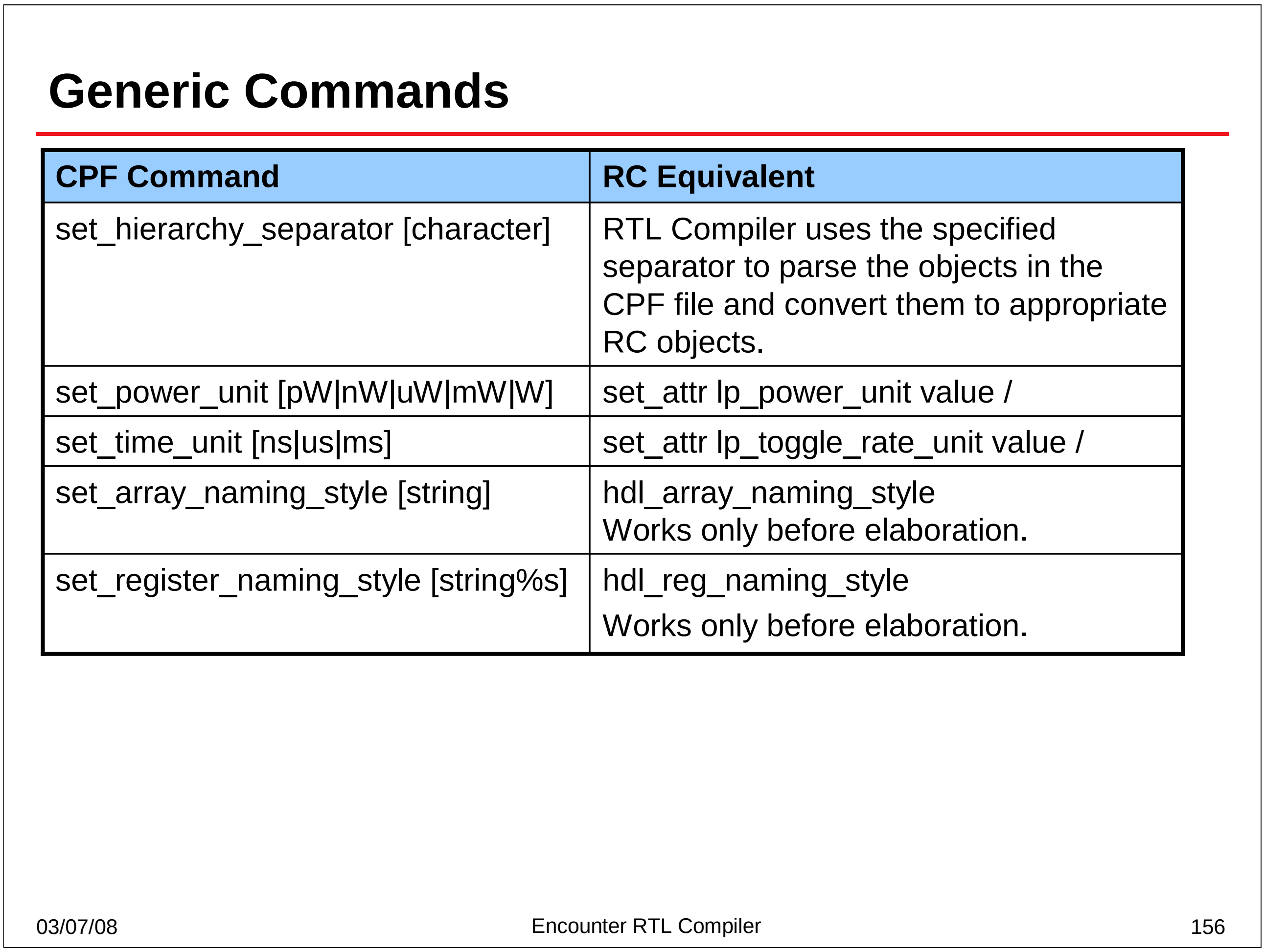
The always-on cells are required in the control logic, which generates the control signals for isolation or state retention cells.

Use this CPF command to define cells to use as always-on cells.

```
define_always_on_cell -cells <cell_list>
```

RC supports only always-on buffers and inverters.

It is necessary to have always-on logic in a switched-off logic block (power domain).



Generic Commands

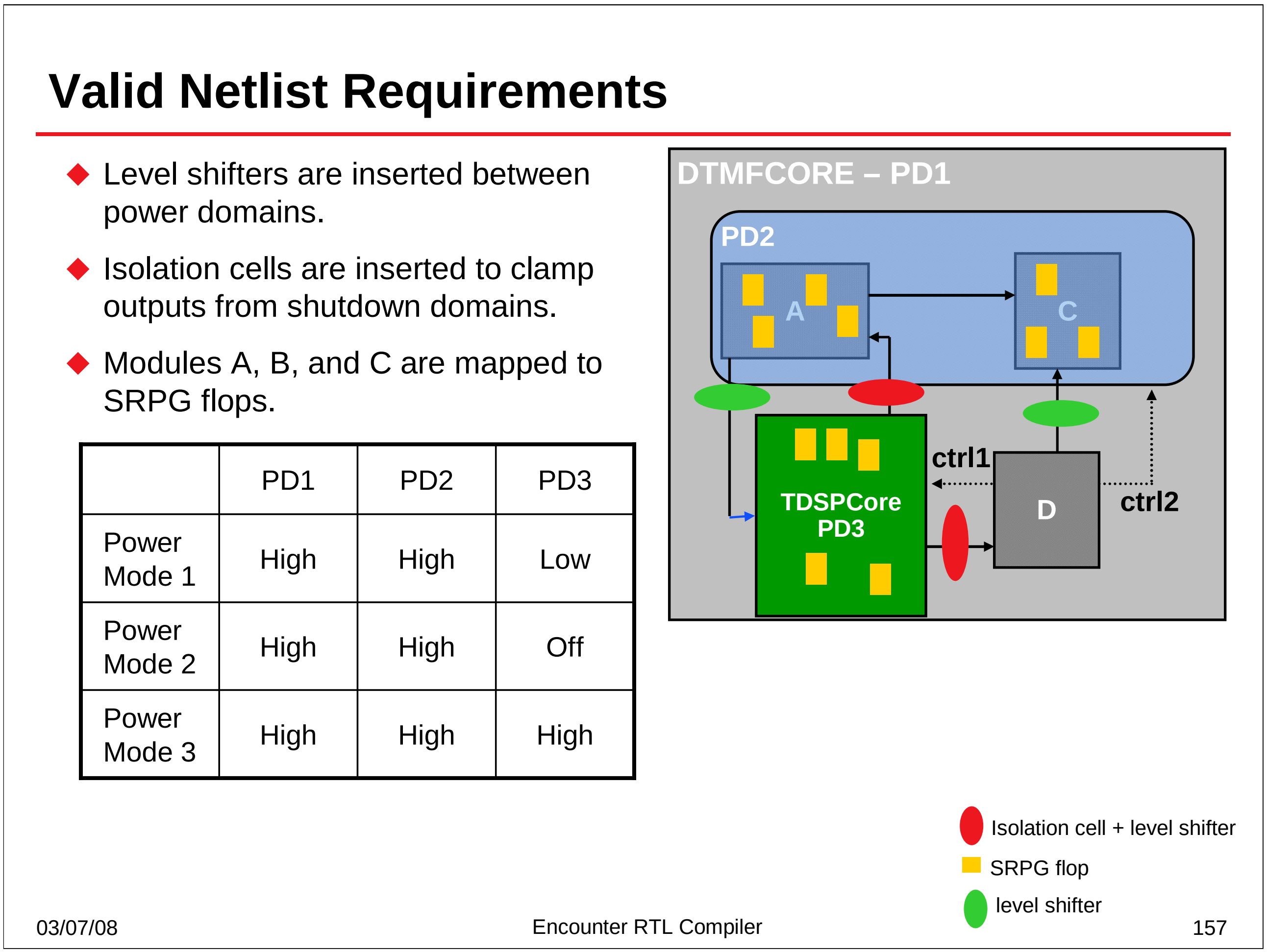
CPF Command	RC Equivalent
set_hierarchy_separator [character]	RTL Compiler uses the specified separator to parse the objects in the CPF file and convert them to appropriate RC objects.
set_power_unit [pW|nW|uW|mW|W]	set_attr lp_power_unit value /
set_time_unit [ns|us|ms]	set_attr lp_toggle_rate_unit value /
set_array_naming_style [string]	hdl_array_naming_style Works only before elaboration.
set_register_naming_style [string%s]	hdl_reg_naming_style Works only before elaboration.

Units

The *time_unit* and *power_unit* are not scope-sensitive. Therefore, only the first command is taken into account.

Naming Styles

RTL Compiler checks to see if the style corresponds to the style set by the *hdl_reg_naming_style* and *hdl_array_naming_style* attributes. If the styles do not correspond, the compiler issues an error because it cannot change the style after the design has been elaborated.



Valid Netlist Requirements

- Level shifters are inserted between power domains.
- Isolation cells are inserted to clamp outputs from shutdown domains.
- Modules A, B, and C are mapped to SRPG flops.

	PD1	PD2	PD3
Power Mode 1	High	High	Low
Power Mode 2	High	High	Off
Power Mode 3	High	High	High

🔴 Isolation cell + level shifter
🟨 SRPG flop
🟢 level shifter

Here is what you can expect after synthesis:

- The shutdown blocks’ registers are retention registers and isolation logic is placed on the outputs of these blocks.
- You can see that simple level shifters are applied to the input to PD2. This is because PD1 is a different voltage level but is not shut down.

The only way to check for proper level-shifter insertion is with Encounter® Conformal® Low Power. Level shifters have no functionality, so they cannot be exhaustively checked by any other tool.

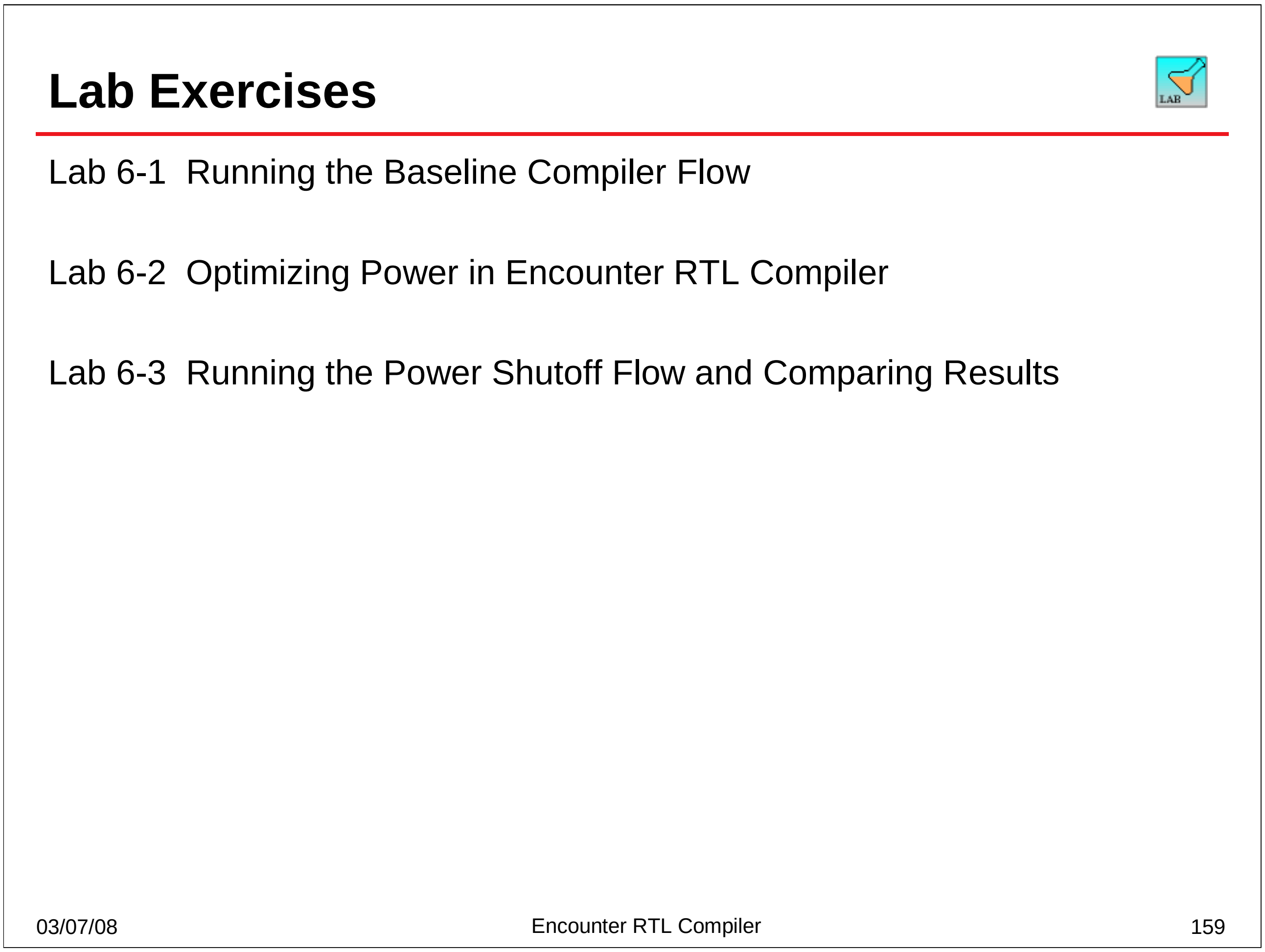
Activity: CPF File

What is a CPF file?

What are some examples of power shutoff implementation rules that need to be specified in a CPF file?

Answer: A CPF is an ASCII file that specifies the power intent used throughout design, verification, and implementation.

Answer: Example rules to specify in a CPF file include PSO cells and SRPG register insertion.

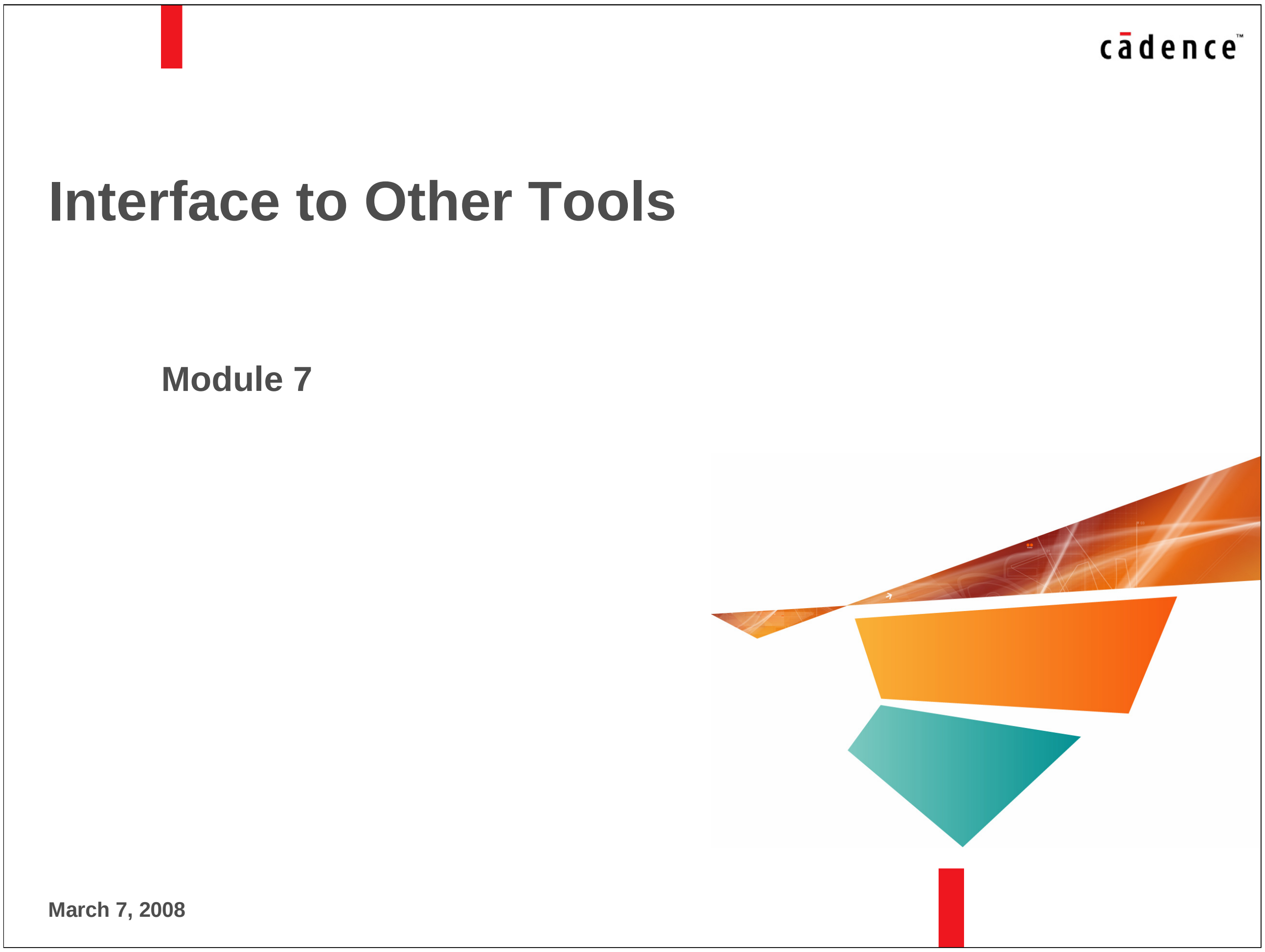


Lab Exercises

Lab 6-1 Running the Baseline Compiler Flow

Lab 6-2 Optimizing Power in Encounter RTL Compiler

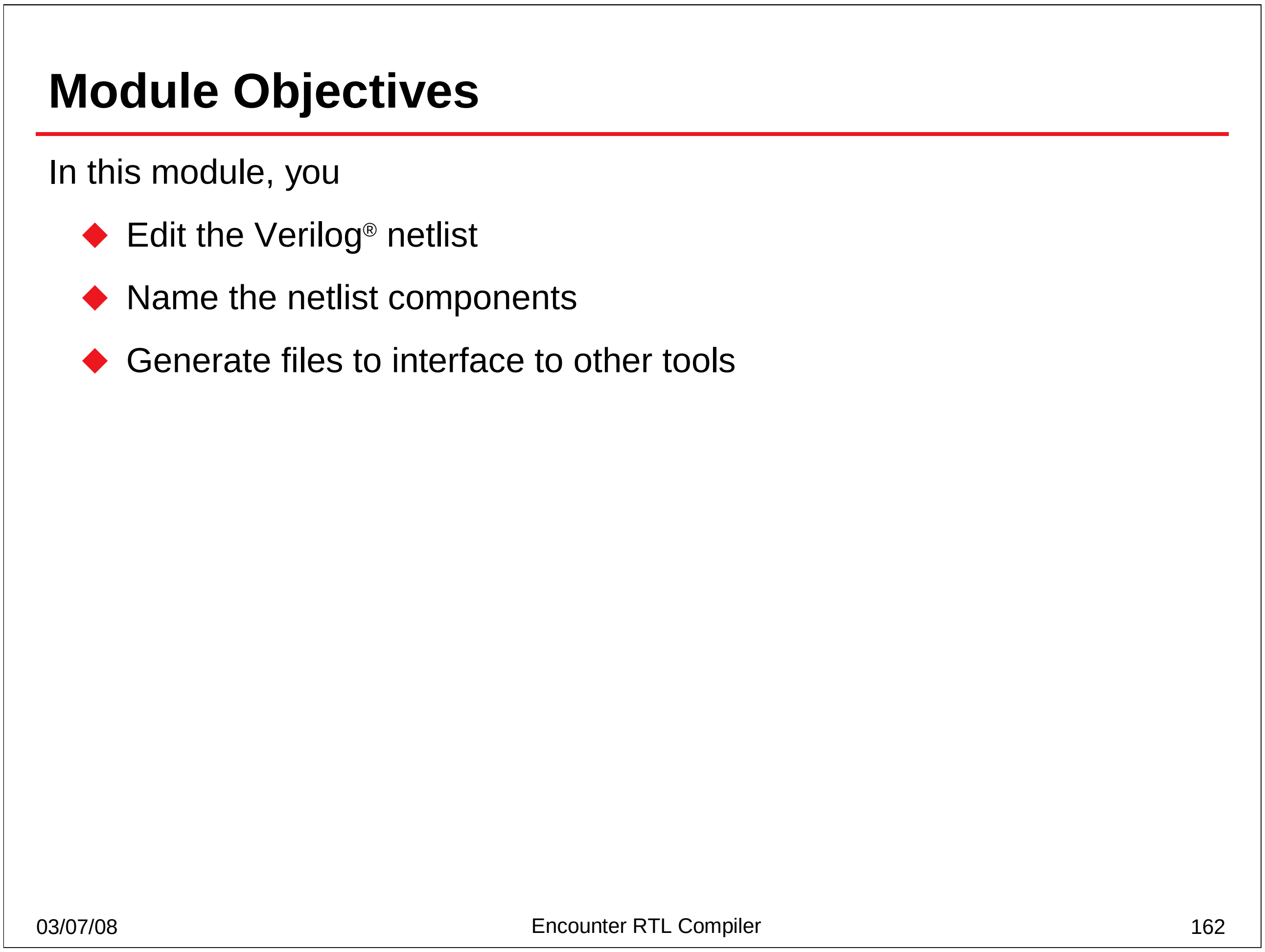
Lab 6-3 Running the Power Shutoff Flow and Comparing Results



Interface to Other Tools

Module 7

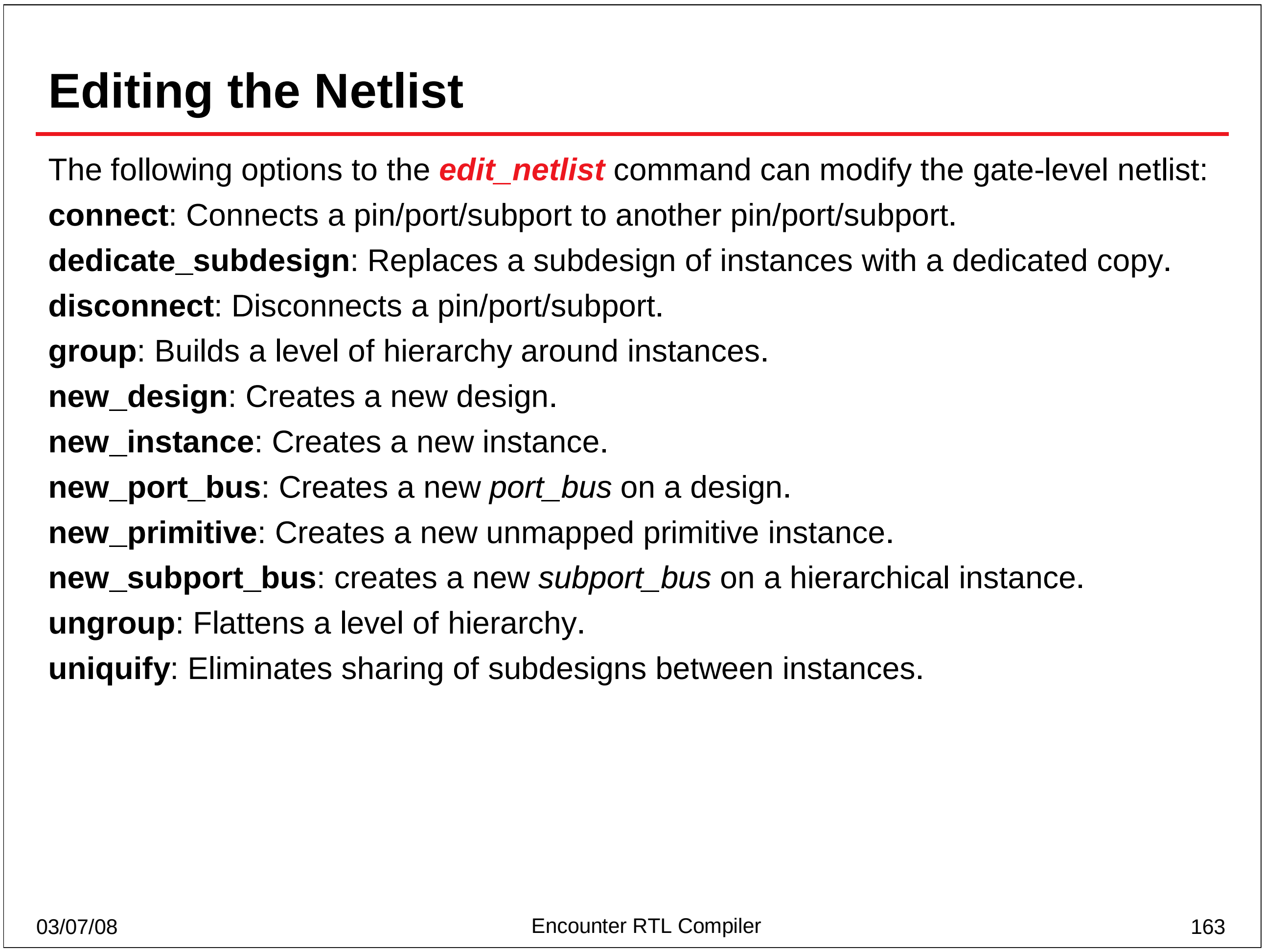
March 7, 2008



Module Objectives

In this module, you

- Edit the Verilog® netlist
- Name the netlist components
- Generate files to interface to other tools



Editing the Netlist

The following options to the ***edit_netlist*** command can modify the gate-level netlist:

connect: Connects a pin/port/subport to another pin/port/subport.

dedicate_subdesign: Replaces a subdesign of instances with a dedicated copy.

disconnect: Disconnects a pin/port/subport.

group: Builds a level of hierarchy around instances.

new_design: Creates a new design.

new_instance: Creates a new instance.

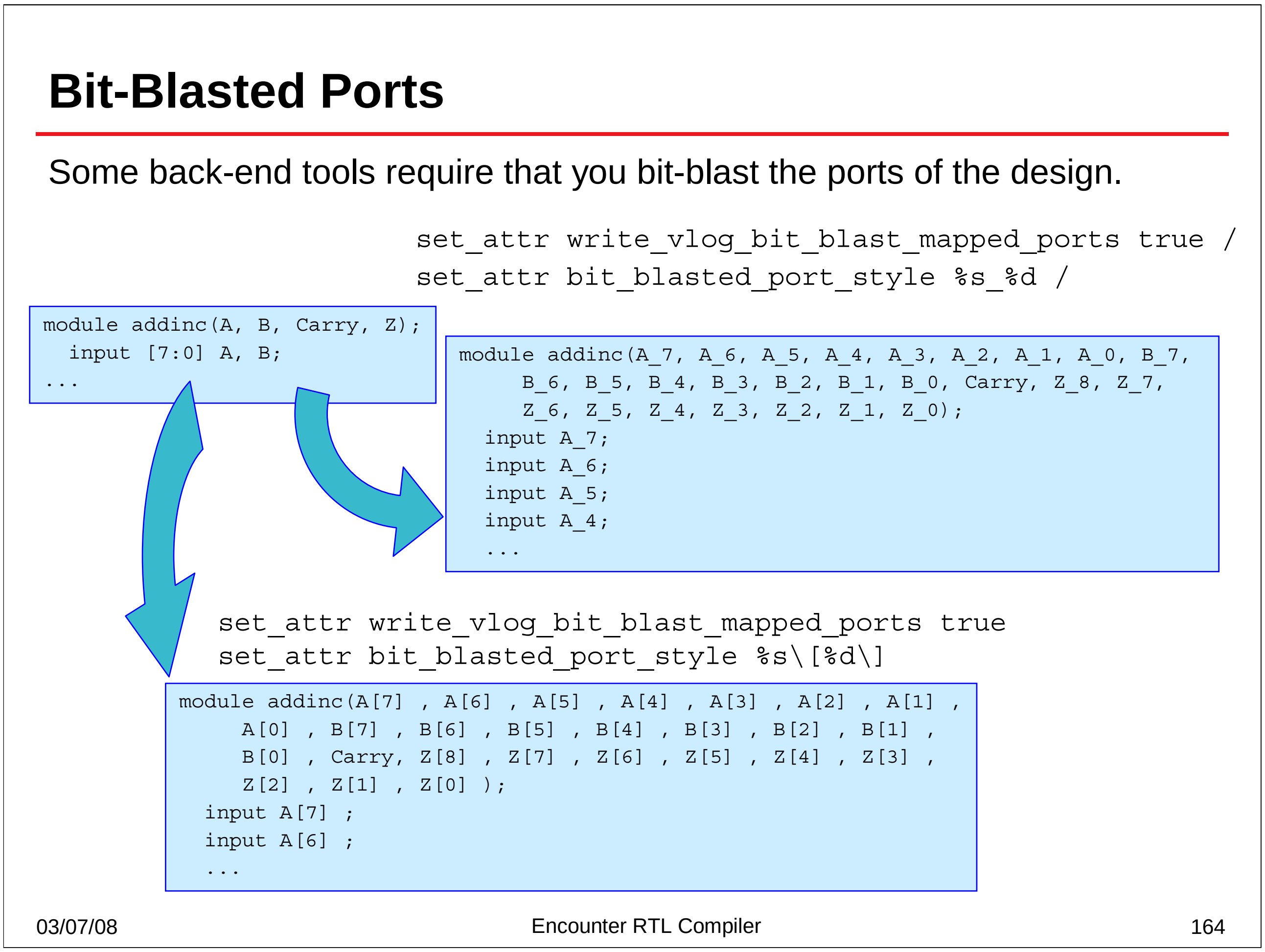
new_port_bus: Creates a new *port_bus* on a design.

new_primitive: Creates a new unmapped primitive instance.

new_subport_bus: creates a new *subport_bus* on a hierarchical instance.

ungroup: Flattens a level of hierarchy.

uniquify: Eliminates sharing of subdesigns between instances.



Bit-Blasted Ports

Some back-end tools require that you bit-blast the ports of the design.

```
set_attr write_vlog_bit_blast_mapped_ports true /
set_attr bit_blasted_port_style %s_%d /
```

```
module addinc(A, B, Carry, Z);
  input [7:0] A, B;
...
```

```
module addinc(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0, B_7,
     B_6, B_5, B_4, B_3, B_2, B_1, B_0, Carry, Z_8, Z_7,
     Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0);
  input A_7;
  input A_6;
  input A_5;
  input A_4;
  ...
```

```
set_attr write_vlog_bit_blast_mapped_ports true
set_attr bit_blasted_port_style %s\[%d\]
```

```
module addinc(A[7] , A[6] , A[5] , A[4] , A[3] , A[2] , A[1] ,
     A[0] , B[7] , B[6] , B[5] , B[4] , B[3] , B[2] , B[1] ,
     B[0] , Carry, Z[8] , Z[7] , Z[6] , Z[5] , Z[4] , Z[3] ,
     Z[2] , Z[1] , Z[0] );
  input A[7] ;
  input A[6] ;
  ...
```

After you set these attributes, you must write out the verilog netlist to get the modified netlist.

Verilog Writer Attributes

Object	Attribute	Default
root	write_vlog_bit_blast_constants	false
root	write_vlog_bit_blast_mapped_ports	false
root	write_vlog_declare_wires	true
root	write_vlog_empty_module_for_logic_abstract	true
root	write_vlog_no_negative_index	false
instance	write_vlog_port_association_style	0
root	write_vlog_top_module_first	false
root	write_vlog_unconnected_port_style	full

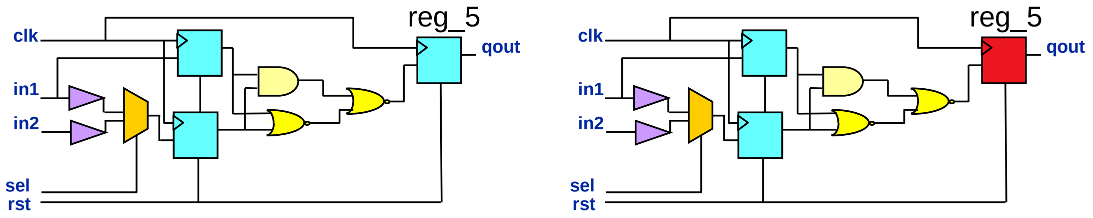
Changing the Instance Library Cell

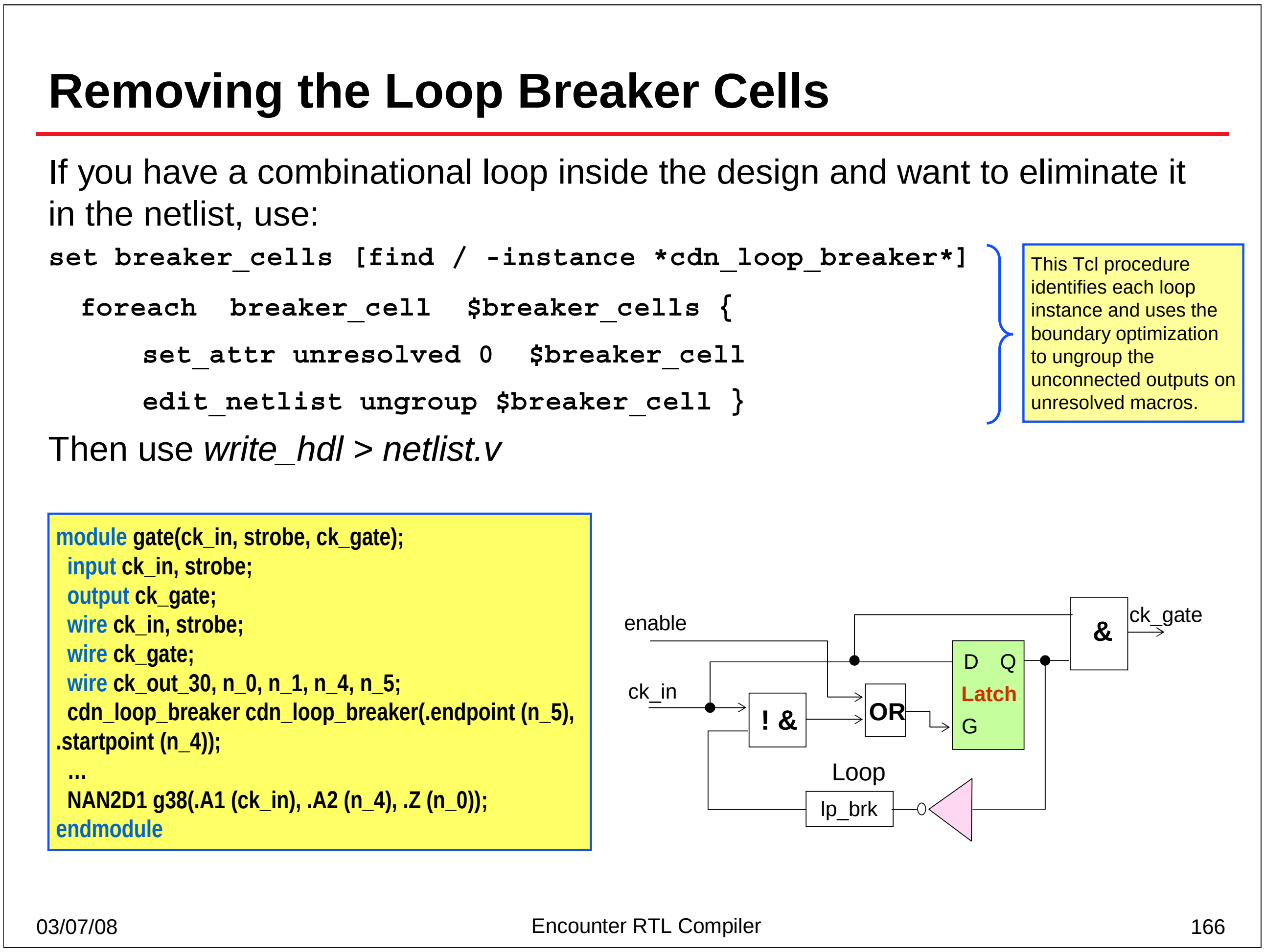
To manually force the instance to have a different library cell, you can use the *libcell* attribute. If you want to replace one cell with another, the pin mappings must be equal.

Example

If you want to use DFFRHQX2 instead of DFFRHQX4 for the reg_5 instance, type the following command:

```
set_attr libcell [find / -libcell DFFRHQX2] \
  /designs/top/instances_hier/I2/instances_seq/reg_5
```





Removing the Loop Breaker Cells

If you have a combinational loop inside the design and want to eliminate it in the netlist, use:

```
set breaker_cells [find / -instance *cdn_loop_breaker*]
  foreach  breaker_cell  $breaker_cells {
      set_attr unresolved 0  $breaker_cell
      edit_netlist ungroup $breaker_cell }
```

This Tcl procedure identifies each loop instance and uses the boundary optimization to ungroup the unconnected outputs on unresolved macros.

Then use *write_hdl > netlist.v*

```
module gate(ck_in, strobe, ck_gate);
  input ck_in, strobe;
  output ck_gate;
  wire ck_in, strobe;
  wire ck_gate;
  wire ck_out_30, n_0, n_1, n_4, n_5;
  cdn_loop_breaker cdn_loop_breaker(.endpoint (n_5),
.startpoint (n_4));
  ...
  NAN2D1 g38(.A1 (ck_in), .A2 (n_4), .Z (n_0));
endmodule
```

During elaboration, combinational loops are replaced by the **cdn_loop_breaker** cells.

By default, the compiler will break combinational loops in a circuit. It does this during the *synthesis -to_generic* or during the *synthesize -to_mapped* commands.

If the information level is set to **7** or higher, then the log file output shows the hierarchical generic gates along the loop.

The disabled timing-arc will be written in the constraints, so that other tools have the loop broken at the same place.

Naming Generated Components

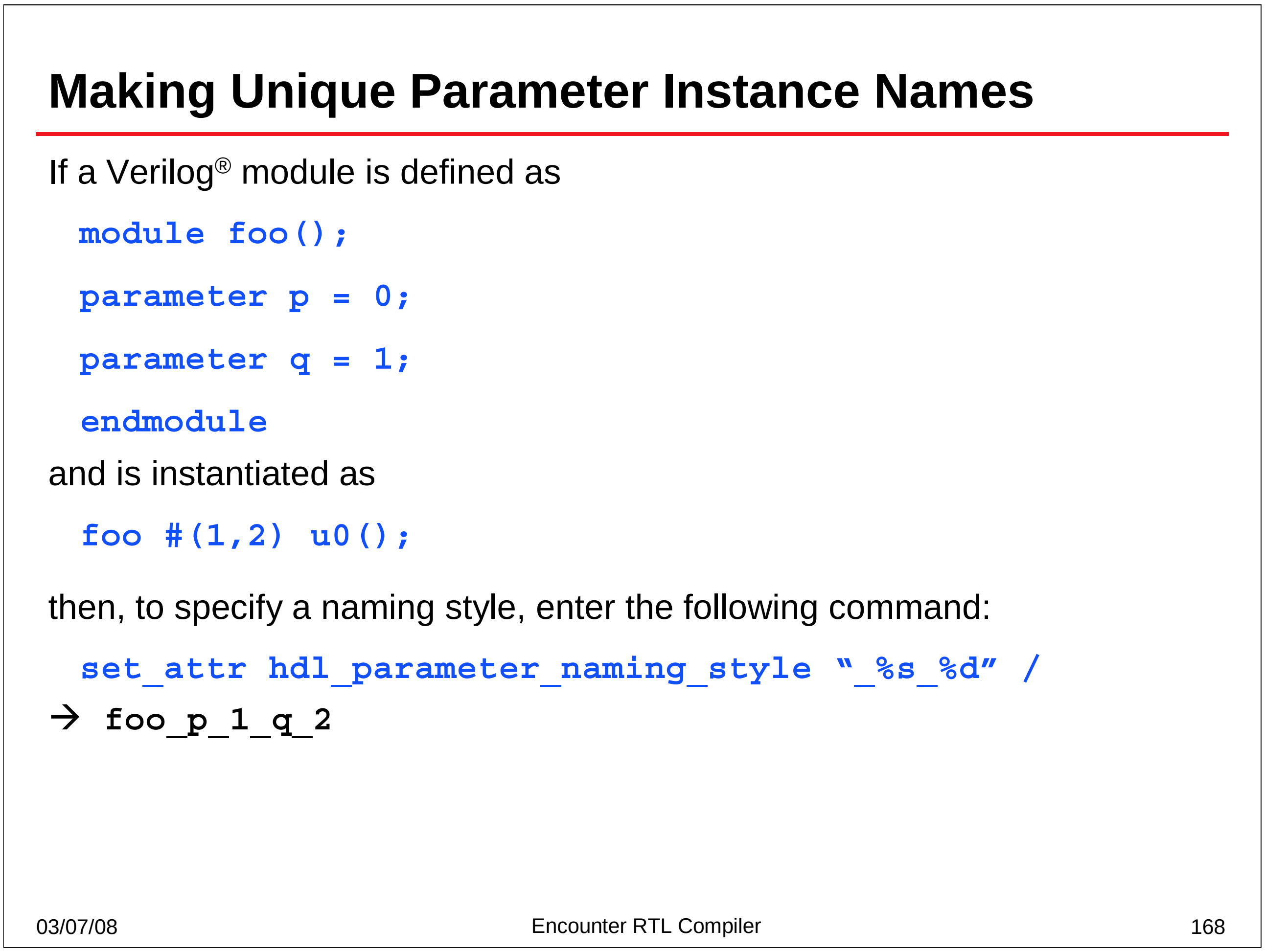
You can set the prefix to the names of the generated modules by using the *gen_module_prefix* attribute.

Example

```
set_attr gen_module_prefix CDN_DP_ /
```

- This command lets you identify these modules easily.
- You can then easily find them later using the command:

```
[find /des* -subdesign CDN_DP*]
```



Making Unique Parameter Instance Names

If a Verilog® module is defined as

```
module foo();
parameter p = 0;
parameter q = 1;
endmodule
```

and is instantiated as

```
foo #(1,2) u0();
```

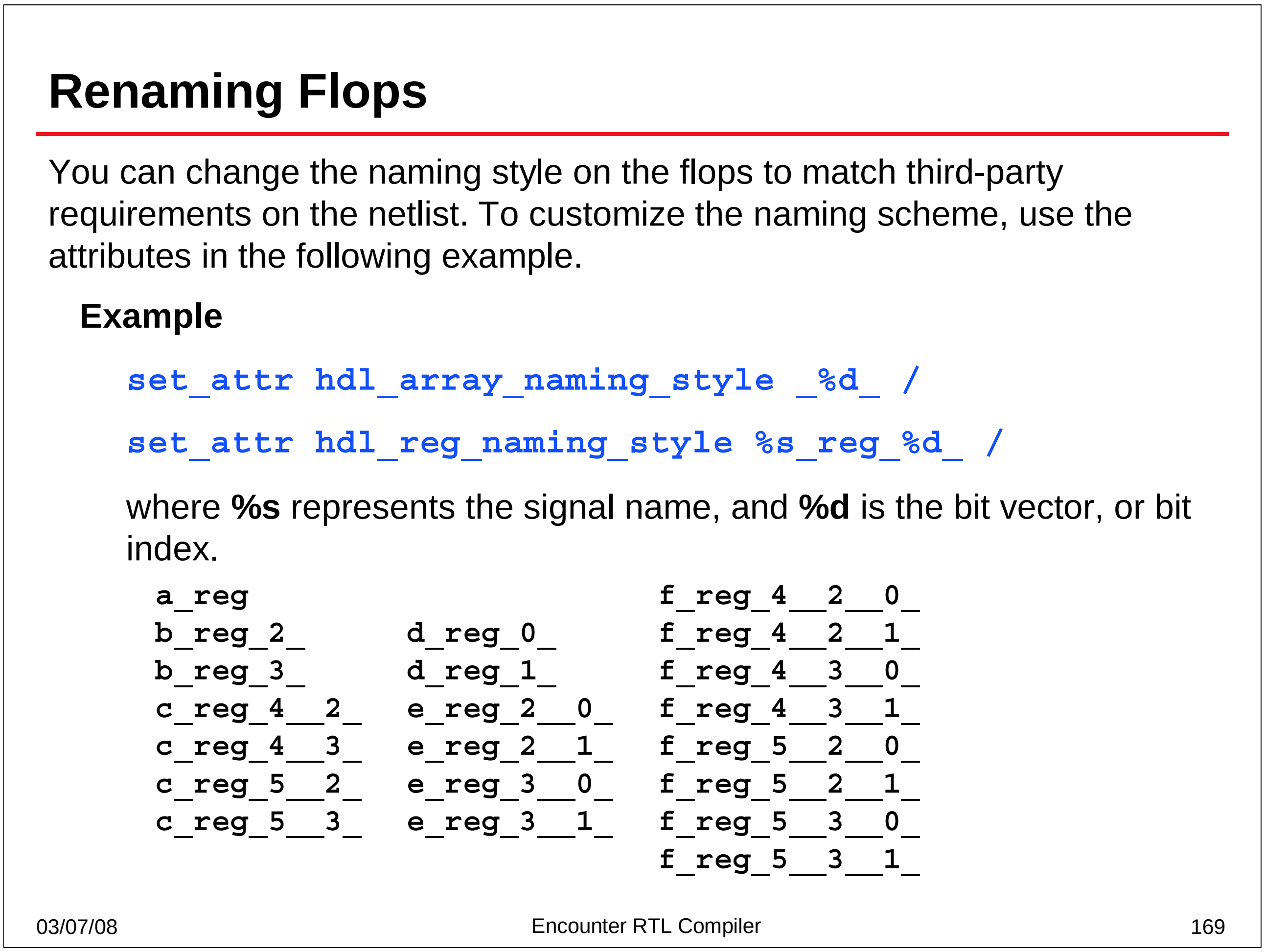
then, to specify a naming style, enter the following command:

```
set_attr hdl_parameter_naming_style “_%s_%d” /
```

→ foo_p_1_q_2

You can match the names generated by Design Compiler with the following variable settings in your script:

- `set hdlin_template_naming_style “%s_%p”`
- `set hdlin_template_parameter_style “%d”`
- `set hdlin_template_separator_style “_”`
- `set hdlin_template_parameter_style_variable “%d”`



Renaming Flops

You can change the naming style on the flops to match third-party requirements on the netlist. To customize the naming scheme, use the attributes in the following example.

Example

```
set_attr hdl_array_naming_style _%d_ /

set_attr hdl_reg_naming_style %s_reg_%d_ /
```

where **%s** represents the signal name, and **%d** is the bit vector, or bit index.

```
a_reg                          f_reg_4__2__0_
b_reg_2_       d_reg_0_        f_reg_4__2__1_
b_reg_3_       d_reg_1_        f_reg_4__3__0_
c_reg_4__2_    e_reg_2__0_     f_reg_4__3__1_
c_reg_4__3_    e_reg_2__1_     f_reg_5__2__0_
c_reg_5__2_    e_reg_3__0_     f_reg_5__2__1_
c_reg_5__3_    e_reg_3__1_     f_reg_5__3__0_
                               f_reg_5__3__1_
```

To match the Design Compiler nomenclature, specify the following:

```
set_attribute hdl_array_naming_style %s_reg\[%d\] /
set_attribute hdl_reg_naming_style %s_reg /
```

Two-dimensional arrays are then represented in the following format in the output netlist.

```
<var_name>_reg_<idx1>_<idx2>
```

Example

```
cout_reg_1_1
```

Changing Object Names

The object names need to match the naming conventions of other tools. The **change_names** command lets you change the names of selected objects, such as nets, busses, and instances.

Syntax

```
change_names [-local] [-net] [-instance] [-design] [-subdesign]
             [-port_bus] [-subport_bus] [-force] [-case_insensitive]
             [-prefix string] [-suffix string] [-restricted string]
             [-replace_char string] [-map string]
```

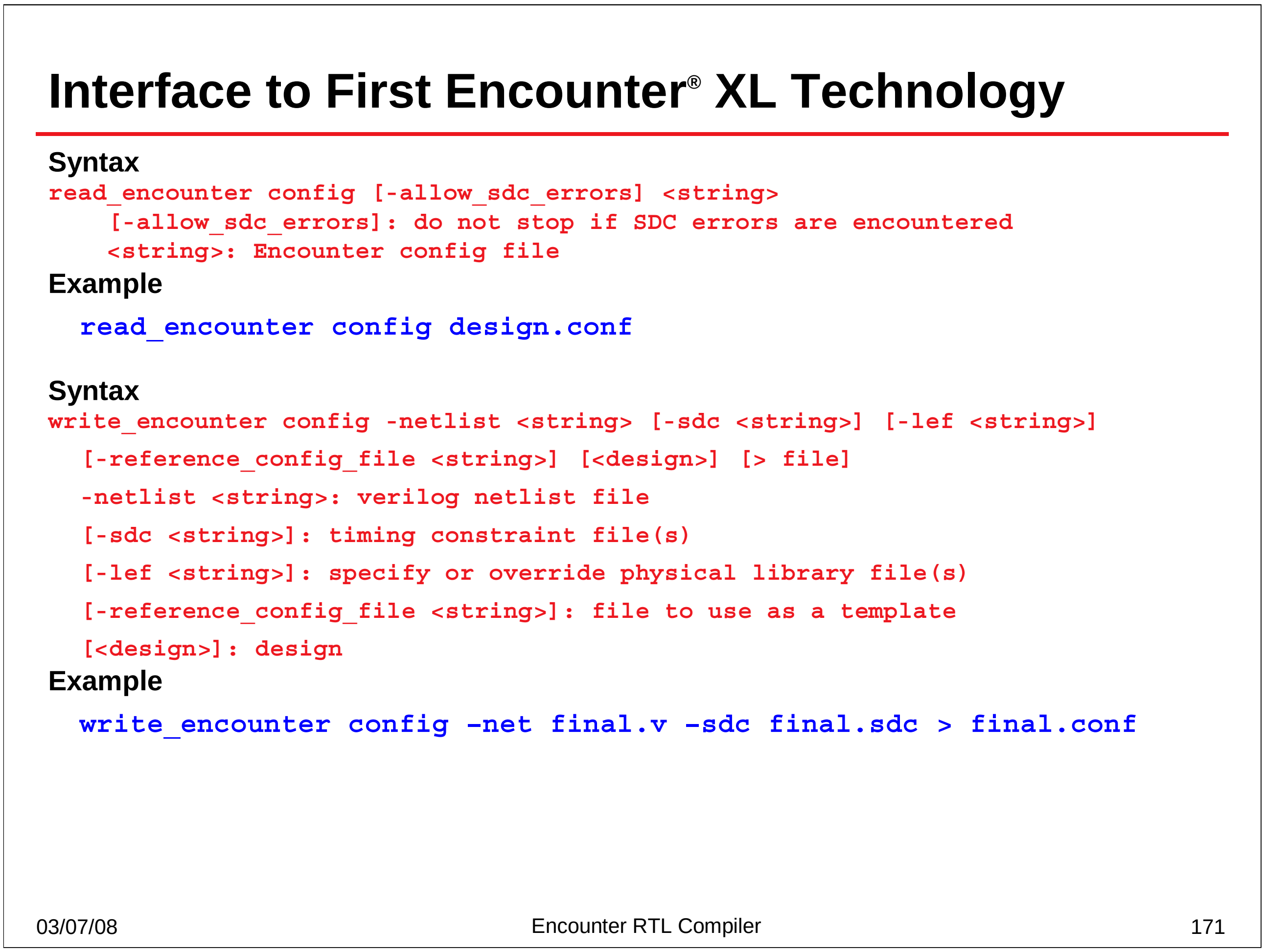
Example

```
change_names -map  {{ "n" "N"} {"_"  "--"}} -net -instance
```

The action is taken on all the instances and nets

- **n** will be replaced with **N**
- _ will be replaced with **- -**

/designs/add1/nets/n_9	/designs/add1/nets/N--9
/designs/a2/instances_hier/add_8_3/	/designs/a2/instances_hier/add--8--3/



Interface to First Encounter® XL Technology

Syntax

```
read_encounter config [-allow_sdc_errors] <string>
    [-allow_sdc_errors]: do not stop if SDC errors are encountered
    <string>: Encounter config file
```

Example

```
  read_encounter config design.conf
```

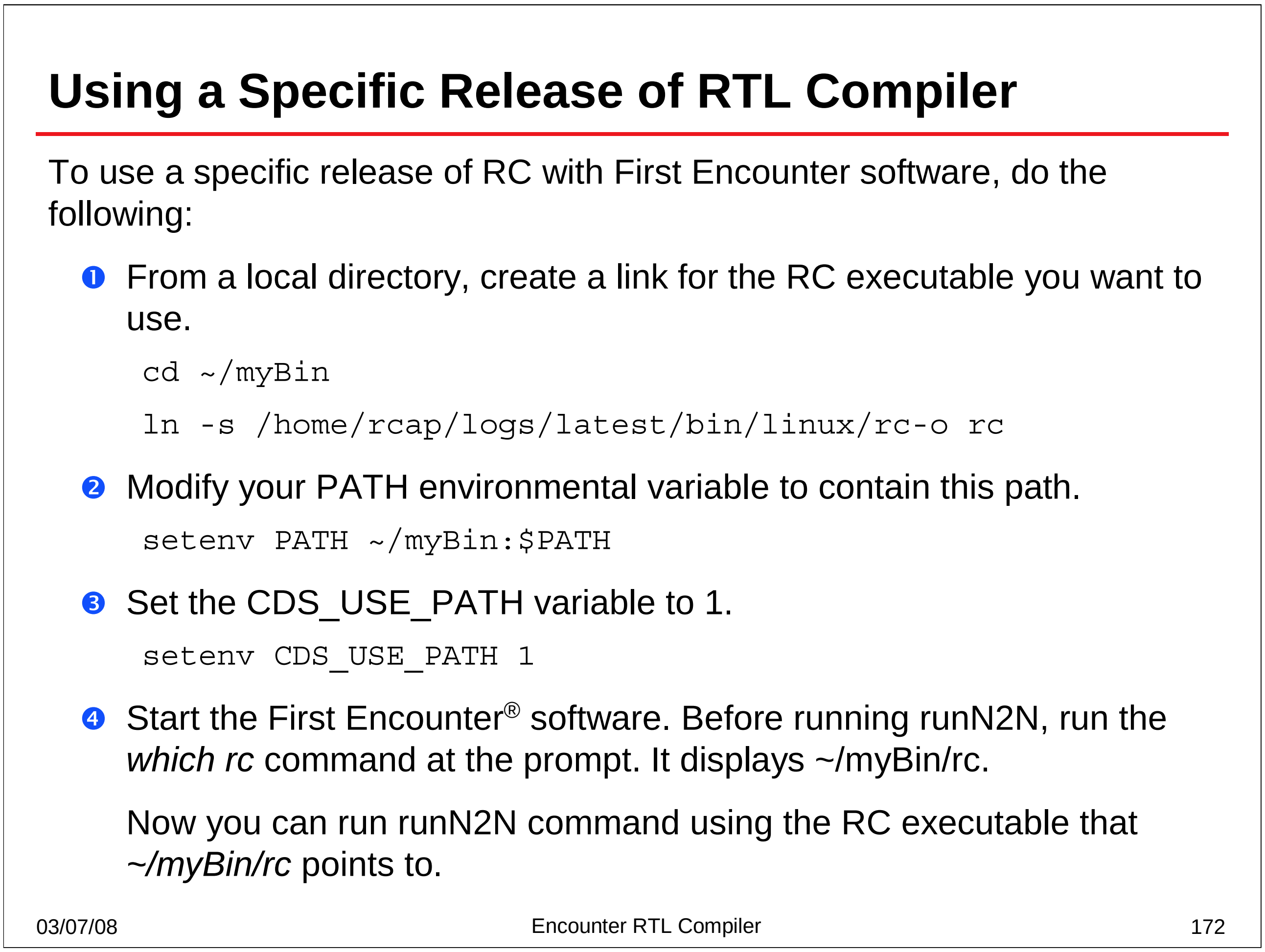
Syntax

```
write_encounter config -netlist <string> [-sdc <string>] [-lef <string>]
  [-reference_config_file <string>] [<design>] [> file]
  -netlist <string>: verilog netlist file
  [-sdc <string>]: timing constraint file(s)
  [-lef <string>]: specify or override physical library file(s)
  [-reference_config_file <string>]: file to use as a template
  [<design>]: design
```

Example

```
  write_encounter config -net final.v -sdc final.sdc > final.conf
```

03/07/08 Encounter RTL Compiler 171



Using a Specific Release of RTL Compiler

To use a specific release of RC with First Encounter software, do the following:

❶ From a local directory, create a link for the RC executable you want to use.

```
cd ~/myBin
ln -s /home/rcap/logs/latest/bin/linux/rc-o rc
```

❷ Modify your PATH environmental variable to contain this path.

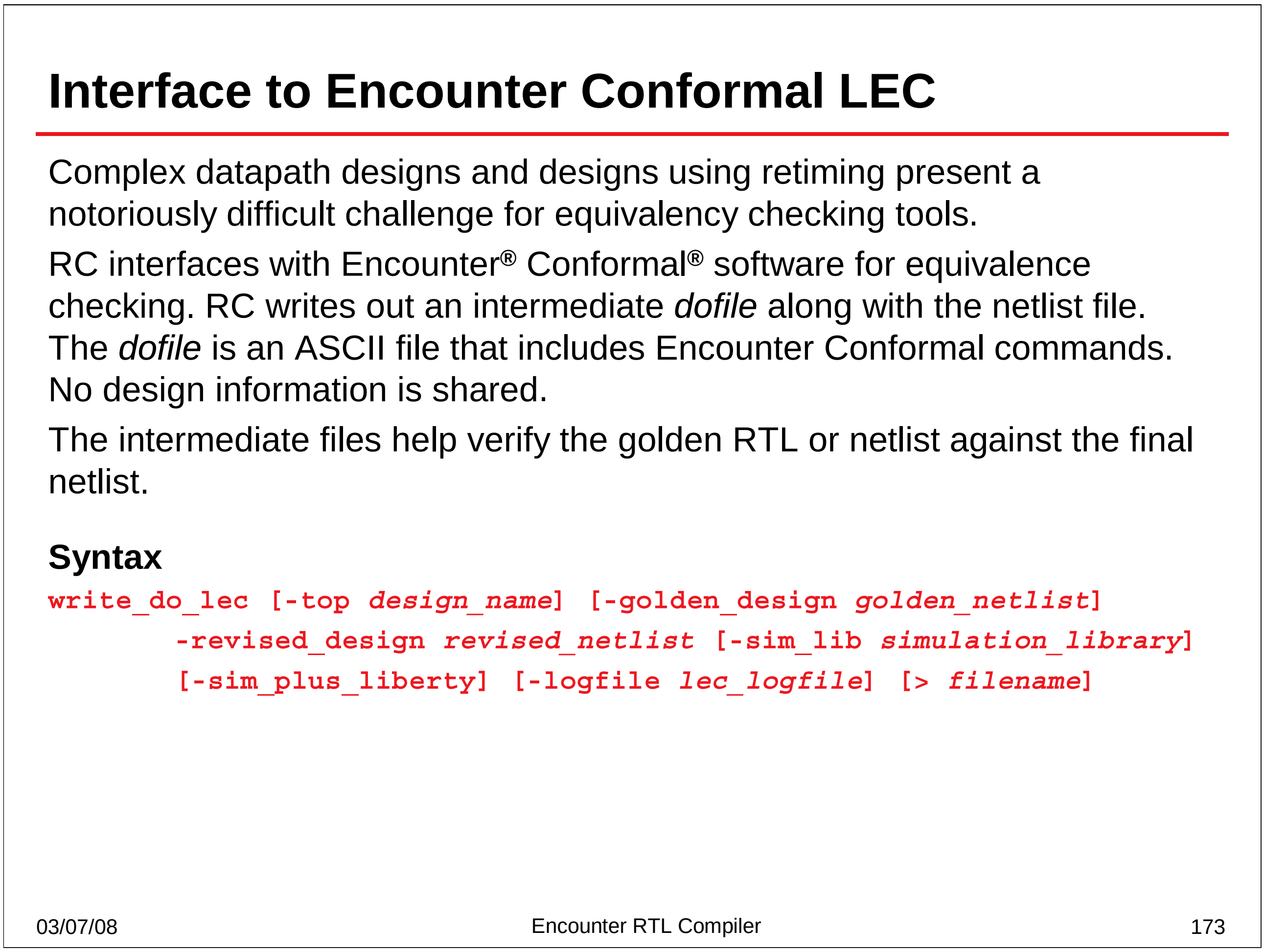
```
setenv PATH ~/myBin:$PATH
```

❸ Set the CDS_USE_PATH variable to 1.

```
setenv CDS_USE_PATH 1
```

❹ Start the First Encounter® software. Before running runN2N, run the *which rc* command at the prompt. It displays ~/myBin/rc.

Now you can run runN2N command using the RC executable that *~/myBin/rc* points to.



Interface to Encounter Conformal LEC

Complex datapath designs and designs using retiming present a notoriously difficult challenge for equivalency checking tools.

RC interfaces with Encounter® Conformal® software for equivalence checking. RC writes out an intermediate *dofile* along with the netlist file. The *dofile* is an ASCII file that includes Encounter Conformal commands. No design information is shared.

The intermediate files help verify the golden RTL or netlist against the final netlist.

Syntax

```
write_do_lec [-top design_name] [-golden_design golden_netlist]
        -revised_design revised_netlist [-sim_lib simulation_library]
        [-sim_plus_liberty] [-logfile lec_logfile] [> filename]
```

The following command flow shows how to generate files for equivalence checking.

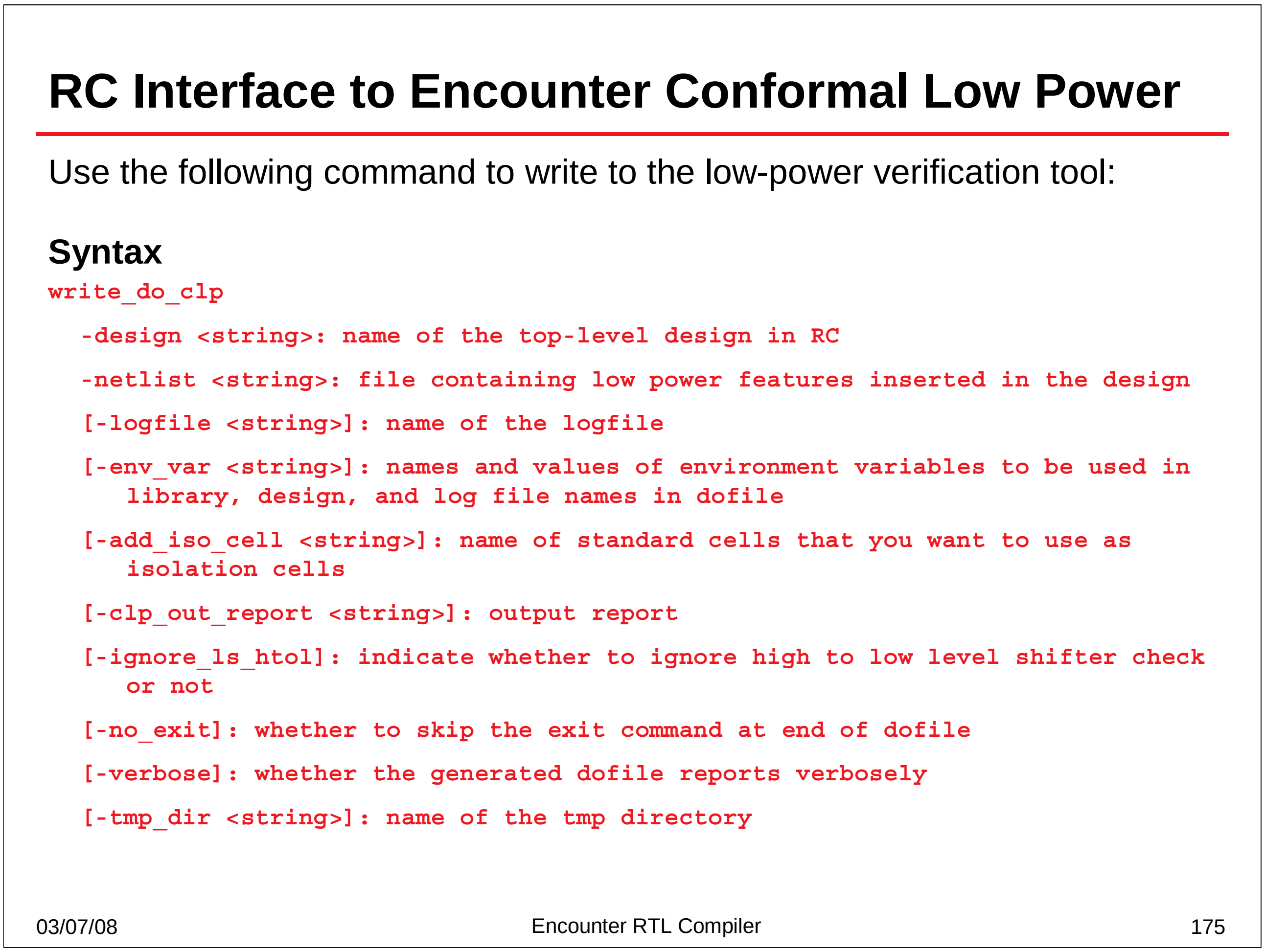
```
set_attr library my_library.lib
read_hdl my_rtl.v
elaborate
read_sdc my_constraints.sdc
synthesize -to_mapped
write_do_lec -golden RTL -revised my_mapped.v -no_exit >
  intermediate.do
ungroup -hierarchical -design top
write_hdl > my_mapped.v
write_do_lec -revised my_final.v -no_exit > final.do
```

Export MSV Designs to Place and Route

To export the necessary files for the Encounter® software, use the following command:

```
write_encounter design [-basename string]
   [-gzip_files] [-reference_config_file config_file]
   [-ignore_scan_chains] [-ignore_msv]
   [-floorplan string] [-lef lef_files] [design ]
```

This command writes files such as the netlist file, the SDC constraint file, templates for the Encounter configuration file, setup file, mode file, scanDEF file, MSV files. You might have to modify the resulting files as needed to fill in the rest of the required information to load it into the Encounter software.



RC Interface to Encounter Conformal Low Power

Use the following command to write to the low-power verification tool:

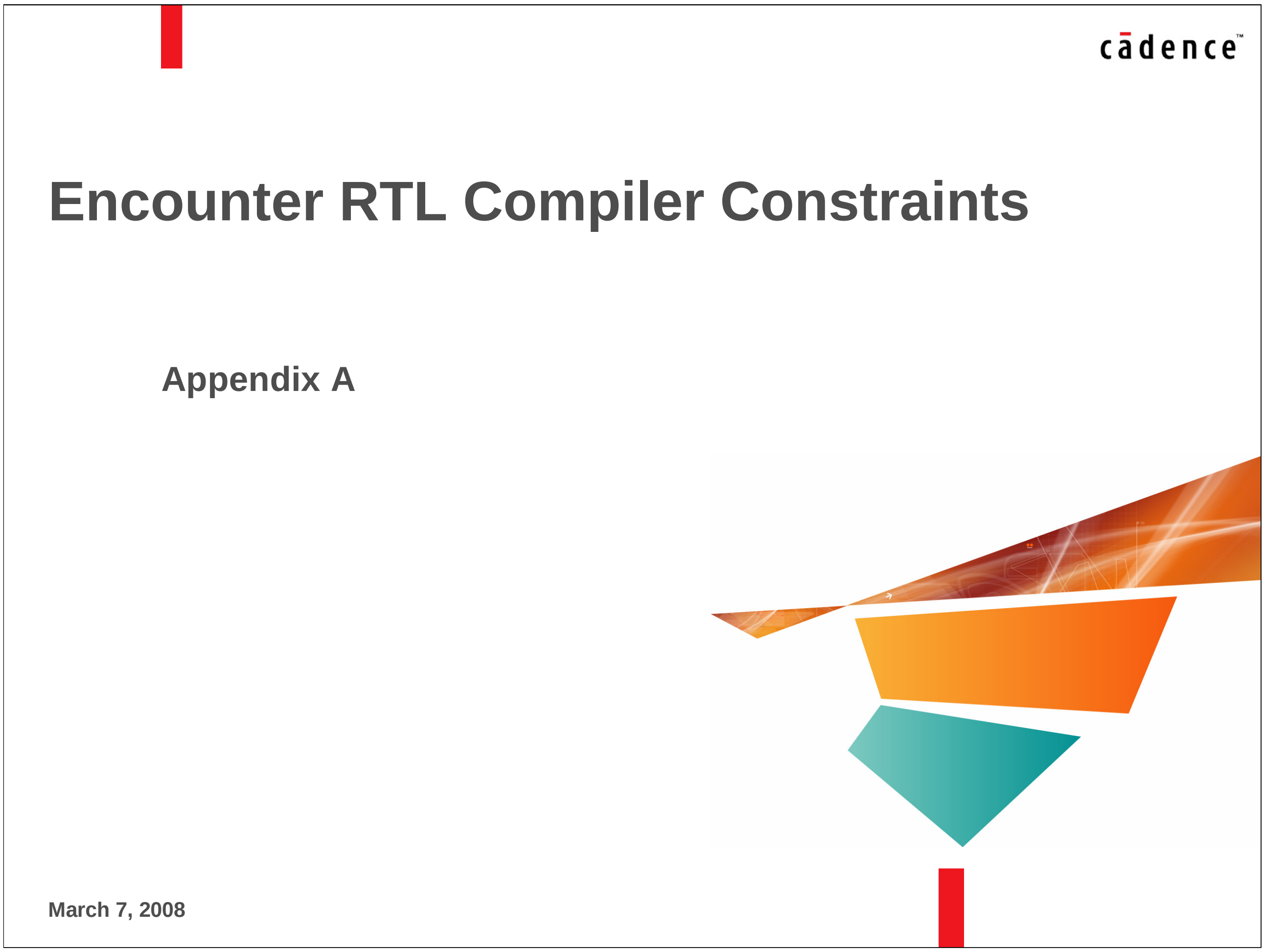
Syntax

```
write_do_clp
  -design <string>: name of the top-level design in RC
  -netlist <string>: file containing low power features inserted in the design
  [-logfile <string>]: name of the logfile
  [-env_var <string>]: names and values of environment variables to be used in
     library, design, and log file names in dofile
  [-add_iso_cell <string>]: name of standard cells that you want to use as
     isolation cells
  [-clp_out_report <string>]: output report
  [-ignore_ls_htol]: indicate whether to ignore high to low level shifter check
     or not
  [-no_exit]: whether to skip the exit command at end of dofile
  [-verbose]: whether the generated dofile reports verbosely
  [-tmp_dir <string>]: name of the tmp directory
```

Lab Exercises

Lab 7-1 Interfacing with Other Tools

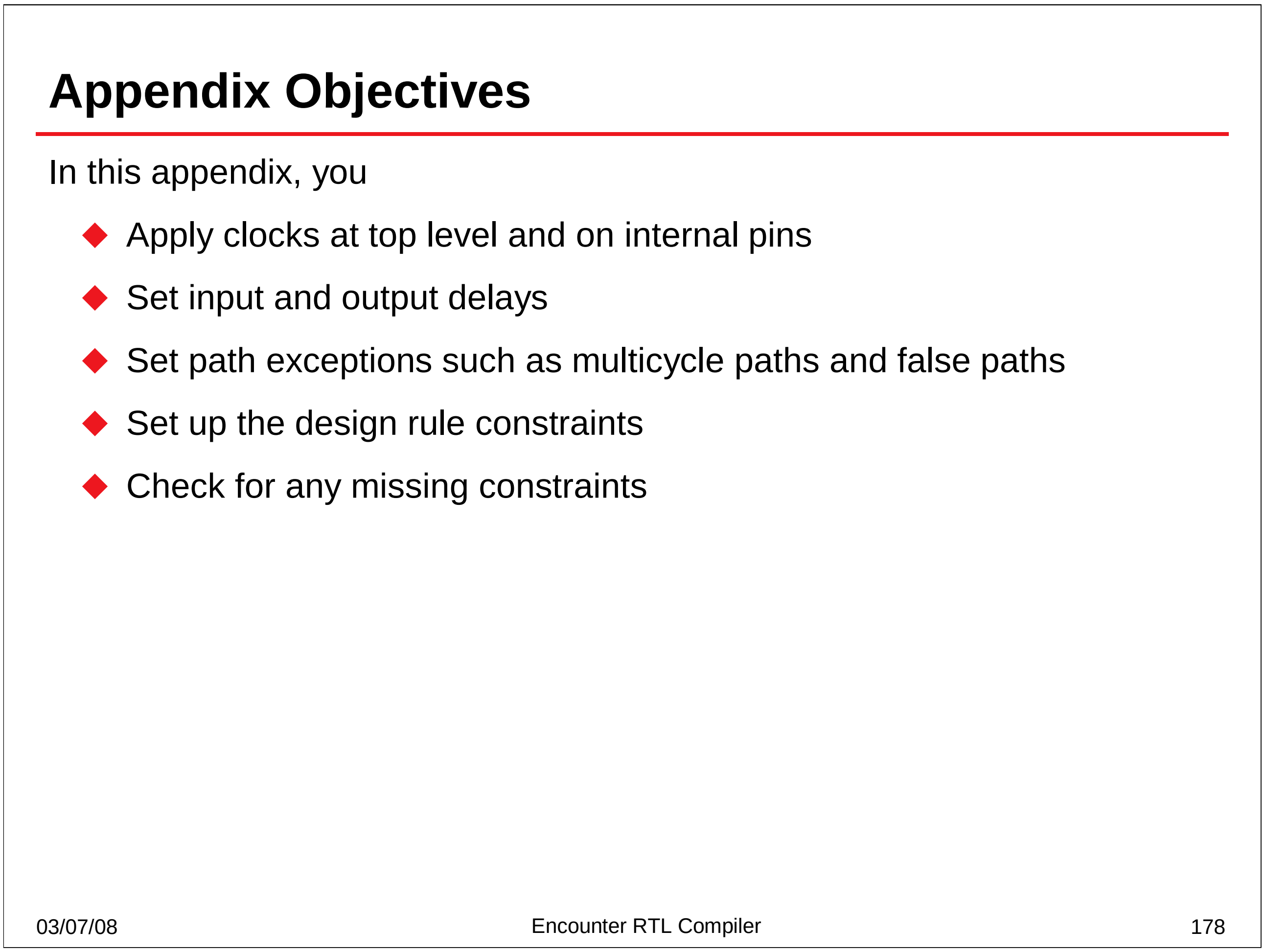
- Changing Names of Design Objects
- Removing Assign Statements from the Netlist
- Controlling the Bit-Blasting of Bus Ports
- Ungrouping the Hierarchy
- Generating Interface Files to Other Tools



Encounter RTL Compiler Constraints

Appendix A

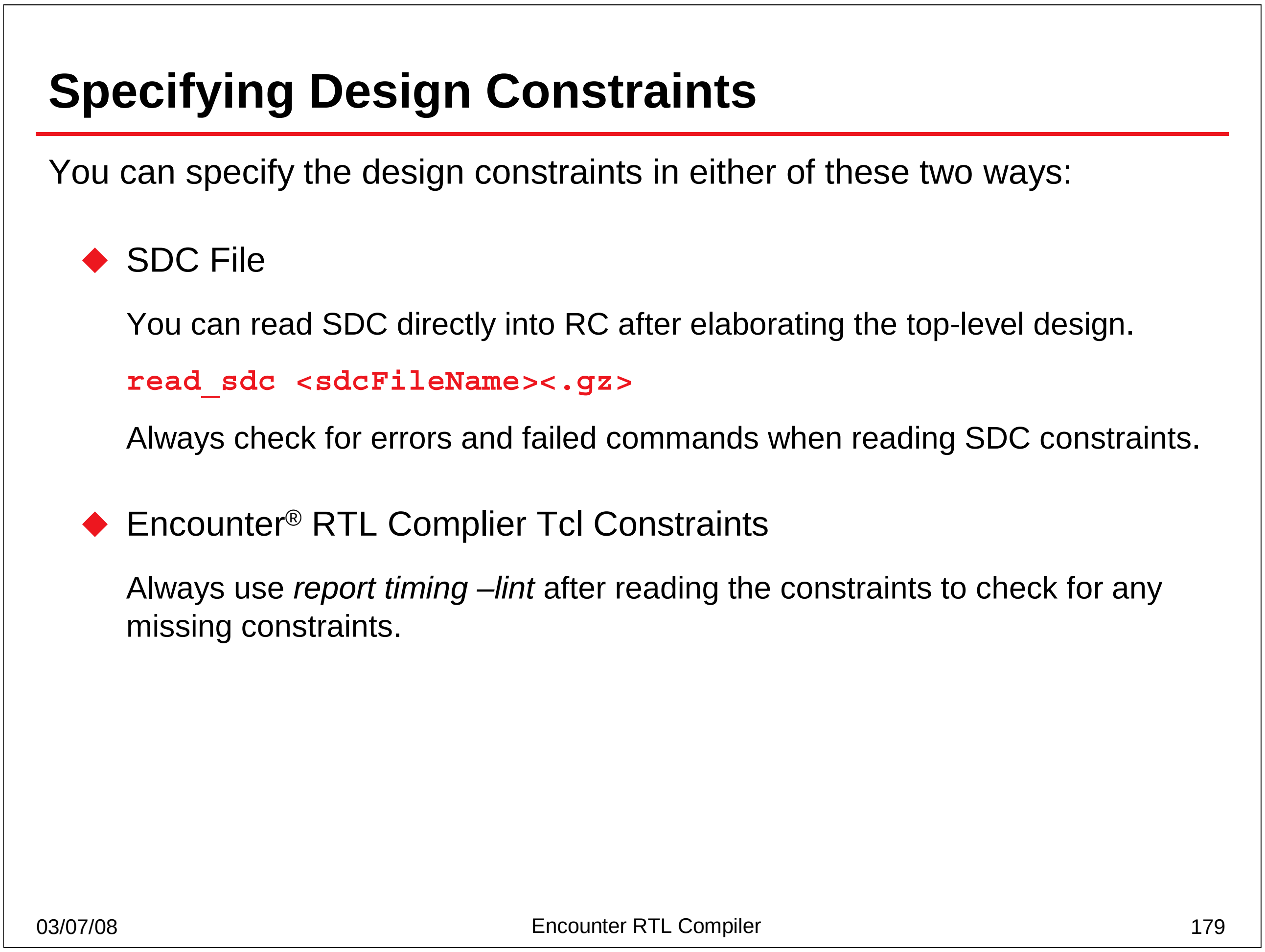
March 7, 2008



Appendix Objectives

In this appendix, you

- Apply clocks at top level and on internal pins
- Set input and output delays
- Set path exceptions such as multicycle paths and false paths
- Set up the design rule constraints
- Check for any missing constraints



Specifying Design Constraints

You can specify the design constraints in either of these two ways:

- SDC File

  You can read SDC directly into RC after elaborating the top-level design.

  ```
  read_sdc <sdcFileName><.gz>
  ```

  Always check for errors and failed commands when reading SDC constraints.

- Encounter® RTL Complier Tcl Constraints

  Always use *report timing –lint* after reading the constraints to check for any missing constraints.

03/07/08 Encounter RTL Compiler 179

Use an SDC file when you import a design from other synthesis environments like BuildGates® Extreme or Design Compiler.

When using an SDC file, the capacitance specified in picofarads *(pF)* (SDC unit) is converted to femtofarads *(fF)* (RC unit) and time specified in *ns* is converted to *ps*.

You can use SDC commands interactively by using the **dc::** as a prefix. But when mixing DC commands and Encounter® RTL Compiler (RC) commands, be very careful with the units of capacitance and delay.

```
dc::set_load [get_attr load slow/INVX1/A] [dc::all_outputs]
```

In this case, the capacitance on all outputs will be off by a factor of 1000 because of the conversion from pF to fF. For example, *get_attr* , which is an RC command returns a load value of *10000 fF* from INVX1/A. The DC command *set_load*, is expecting loads in DC units (*pF*), and sets a load of *10000 pF* on all the outputs.

Instead, use separate commands with conversion factor included.

```
set loadIV [ expr [get_attr load slow/INVX1/A]/1000]
dc:: set_load loadIV [dc::all_outputs]
```

Remember to use **dc::** with every command, even if used recursively (command within a command).

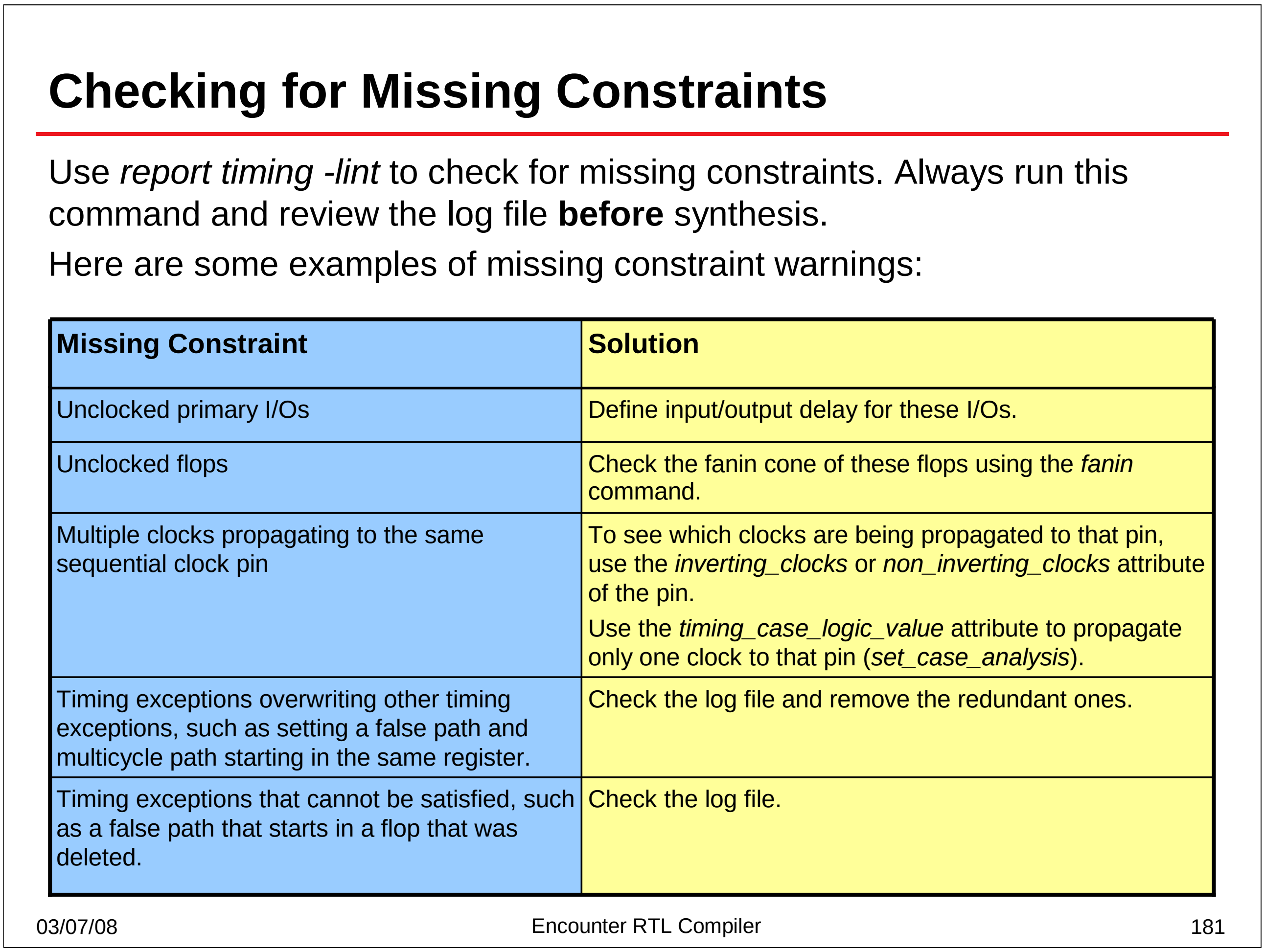
Defining Clock Domains

When the clocks are defined in different clock domains, then

- The clocks will not be considered as synchronous.
  (The timing report will show *async* for the clocks.)
- The compiler will place functional false paths between the clock domains automatically.

```
define_clock -period 10000 -name 100MHz -domain clocka
   [find / -port clka]

define_clock -period 20000 -name 50MHz -domain clockb
   [find / -port clkb]
```

There is no SDC equivalent command for setting clock domains. You have to specify functional false paths between asynchronous clocks in your design.



Checking for Missing Constraints

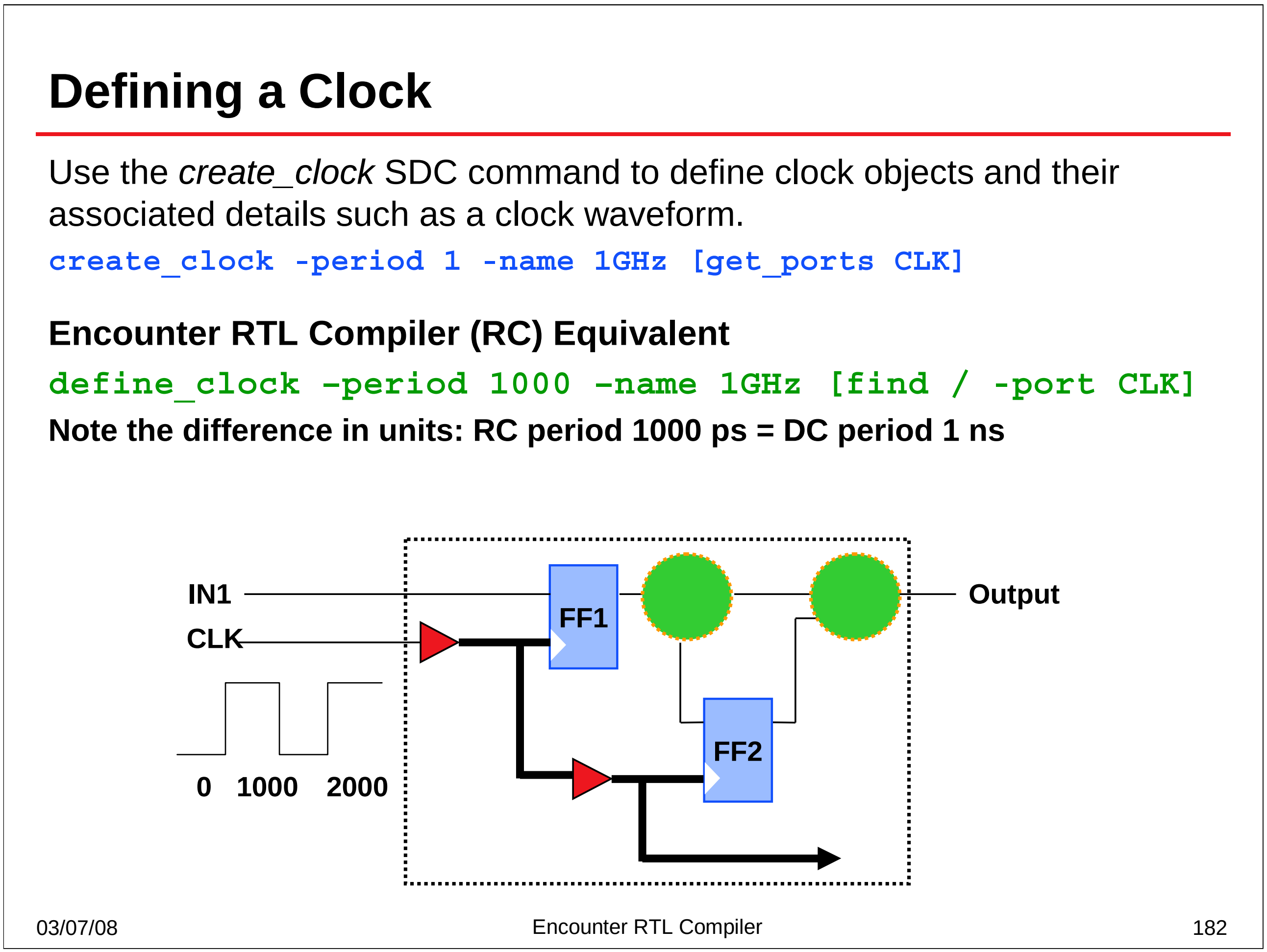
Use *report timing -lint* to check for missing constraints. Always run this command and review the log file **before** synthesis.

Here are some examples of missing constraint warnings:

Missing Constraint	**Solution**
Unclocked primary I/Os	Define input/output delay for these I/Os.
Unclocked flops	Check the fanin cone of these flops using the *fanin* command.
Multiple clocks propagating to the same sequential clock pin	To see which clocks are being propagated to that pin, use the *inverting_clocks* or *non_inverting_clocks* attribute of the pin. Use the *timing_case_logic_value* attribute to propagate only one clock to that pin (*set_case_analysis*).
Timing exceptions overwriting other timing exceptions, such as setting a false path and multicycle path starting in the same register.	Check the log file and remove the redundant ones.
Timing exceptions that cannot be satisfied, such as a false path that starts in a flop that was deleted.	Check the log file.

03/07/08 Encounter RTL Compiler 181

Use *report timing -lint* after elaborating the design and reading the SDC file to generate a detailed timing problem report.



Defining a Clock

Use the *create_clock* SDC command to define clock objects and their associated details such as a clock waveform.

```
create_clock -period 1 -name 1GHz [get_ports CLK]
```

Encounter RTL Compiler (RC) Equivalent

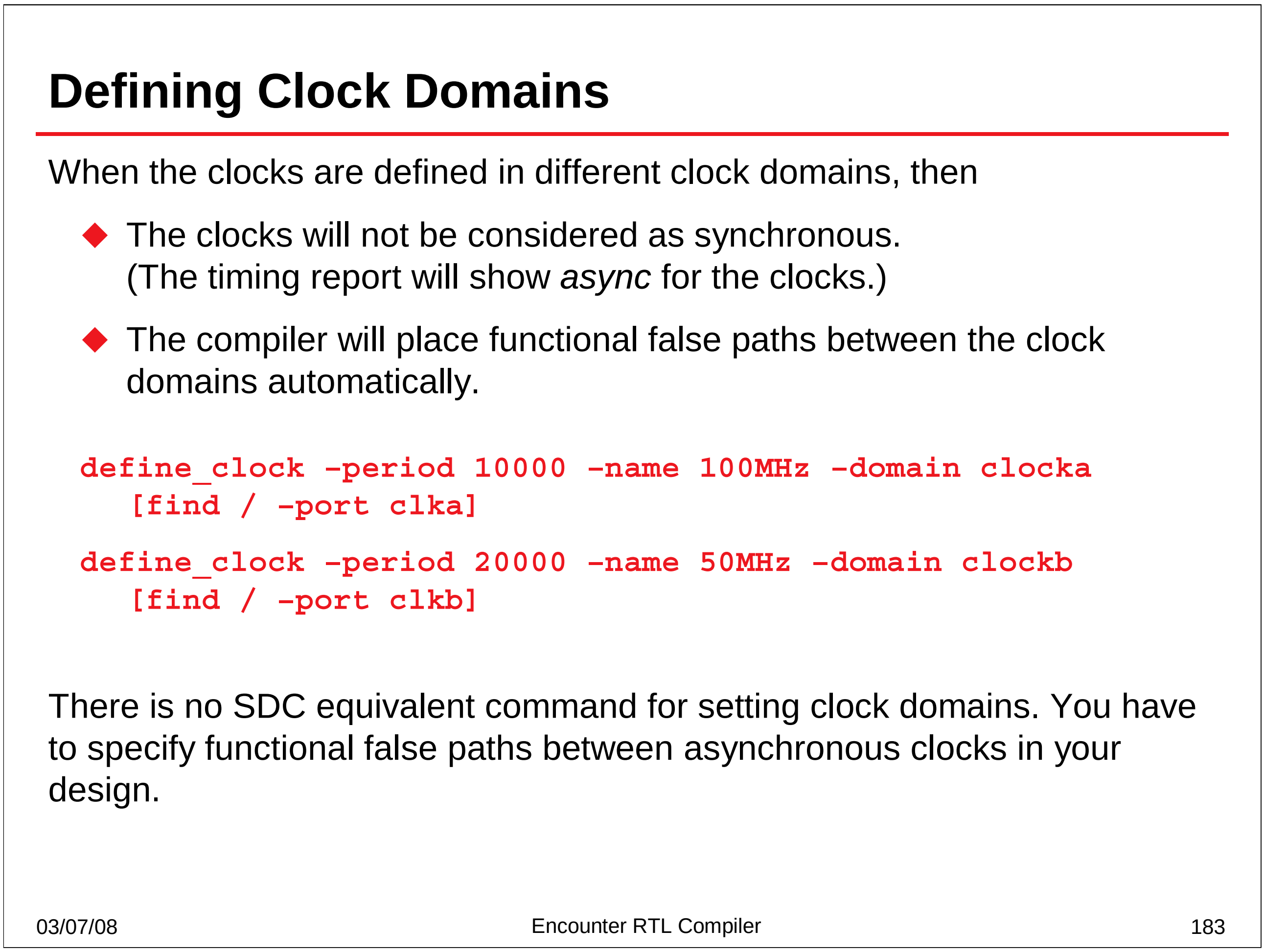
```
define_clock -period 1000 -name 1GHz [find / -port CLK]
```

Note the difference in units: RC period 1000 ps = DC period 1 ns

Syntax

```
define_clock -name <string> [-domain <string>] -period <number>
  [-divide_period <number>] [-rise <number>]
  [-divide_rise <number>] [-fall <number>]
  [-divide_fall <number>] [-design <block_name>]
  [(pin/port)*]
```

`-name`	clock name
`-domain`	name of the clock domain (default is ‘domain_1’)
`-period`	period interval in **picoseconds**
`-divide_period`	clock periods per interval (default is 1)
`-rise`	rise period fraction numerator (default is 0)
`-divide_rise`	rise period fraction denominator (default is 100)
`-fall`	fall period fraction numerator (default is 50)
`-divide_fall`	fall period fraction denominator (default is 100)
`-design`	top-block (required if there are multiple top designs)
`pin/port`	source clock objects



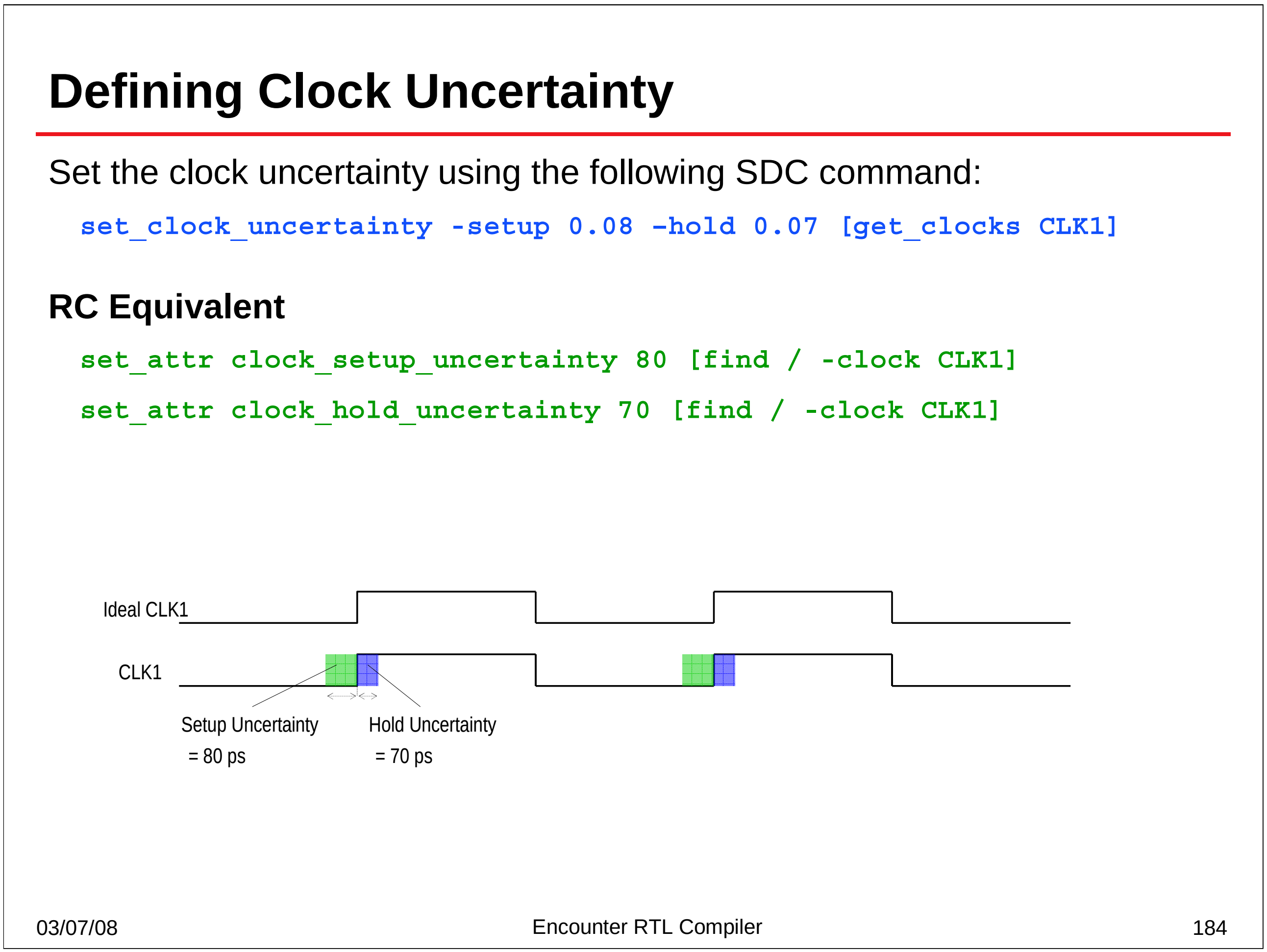
Defining Clock Domains

When the clocks are defined in different clock domains, then

- The clocks will not be considered as synchronous.
  (The timing report will show *async* for the clocks.)
- The compiler will place functional false paths between the clock domains automatically.

```
define_clock -period 10000 -name 100MHz -domain clocka
   [find / -port clka]
define_clock -period 20000 -name 50MHz -domain clockb
   [find / -port clkb]
```

There is no SDC equivalent command for setting clock domains. You have to specify functional false paths between asynchronous clocks in your design.



Defining Clock Uncertainty

Set the clock uncertainty using the following SDC command:

```
set_clock_uncertainty -setup 0.08 -hold 0.07 [get_clocks CLK1]
```

RC Equivalent

```
set_attr clock_setup_uncertainty 80 [find / -clock CLK1]
set_attr clock_hold_uncertainty 70 [find / -clock CLK1]
```

03/07/08 Encounter RTL Compiler 184

- In a single clock domain, uncertainty specifies setup or hold uncertainties to all paths to the endpoint.
- Interclock uncertainty specifies skew between various clock domains.
- Set the uncertainty value to the worst skew expected to the endpoint or to the worst skew between the clock domains.

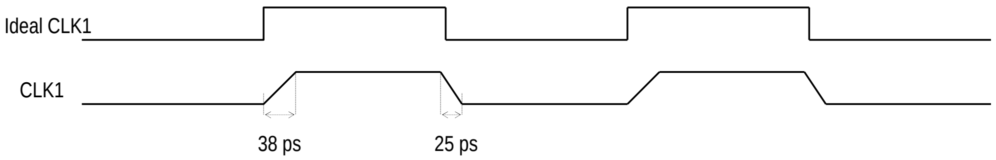
Setting Clock Slew

To set the slew (transition) times in ideal mode, use the following SDC commands:

```
set_clock_transition 0.038 -rise [get_clocks CLK1]
set_clock_transition 0.025 -fall [get_clocks CLK1]
```

RC Equivalent

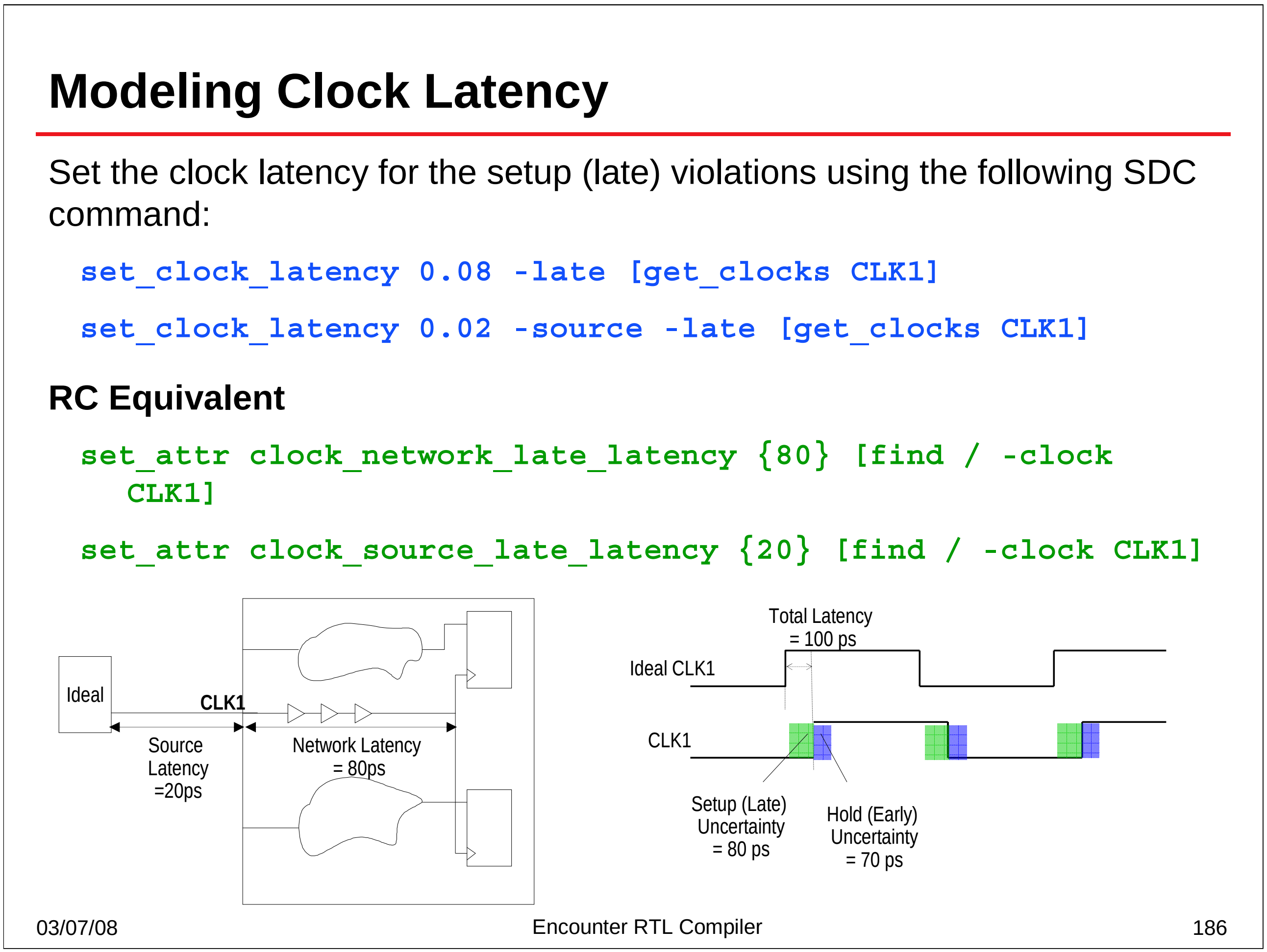
```
set_attribute slew {0 0 38 25} [find / -clock CLK1]
```



Clock slew (transition) is the time a clock takes for its value to change from controlling to noncontrolling or vice-versa.

Usually the *min_rise* and *max_rise* are set to the same value, and *min_fall* and *max_fall* are set to the same value, but you can differentiate them based on the information from your library.

After the clock is in the propagated mode, these values are replaced by the actual values at each pin/port.



Modeling Clock Latency

Set the clock latency for the setup (late) violations using the following SDC command:

```
set_clock_latency 0.08 -late [get_clocks CLK1]
set_clock_latency 0.02 -source -late [get_clocks CLK1]
```

RC Equivalent

```
set_attr clock_network_late_latency {80} [find / -clock
   CLK1]
set_attr clock_source_late_latency {20} [find / -clock CLK1]
```

Ideal
CLK1
Source
Latency
=20ps
Network Latency
= 80ps
Total Latency
= 100 ps
Ideal CLK1
CLK1
Setup (Late)
Uncertainty
= 80 ps
Hold (Early)
Uncertainty
= 70 ps

Clock latency is the time taken by a clock signal to propagate from the clock definition point to a register clock pin. It is also known as insertion delay.

There are two types of latency:

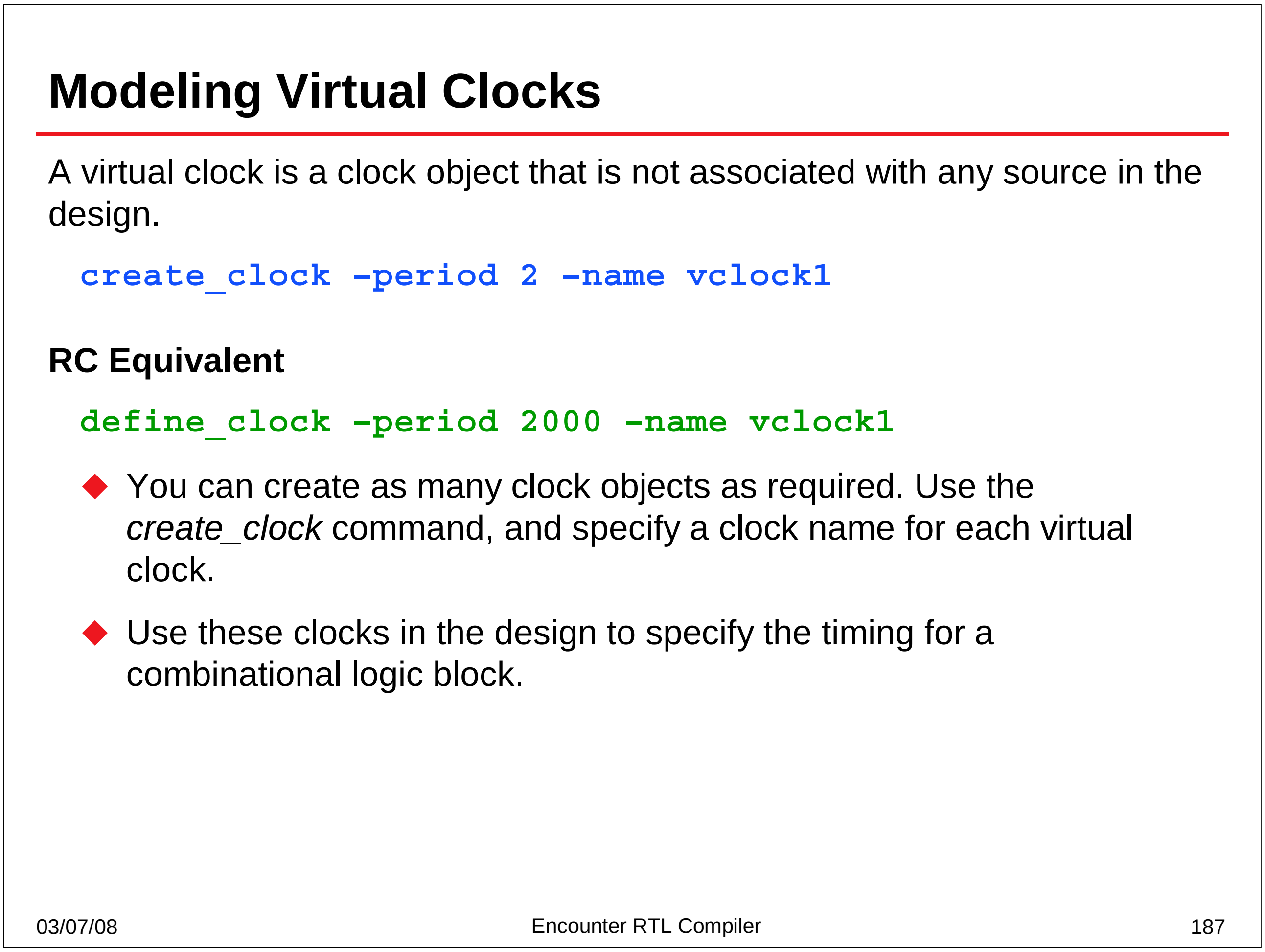
- Source Latency: Clock source to the clock port in the design.
- Network Latency: Clock port to the registers (clock tree delay).

The latency at a register clock pin is the sum of clock source latency and clock network latency.

The value of each delay can be represented as the following four numbers:

```
{min_rise min_fall max_rise max_fall}
```

Usually the *min_rise* and *max_rise* are set to the same value, and *min_fall* and *max_fall* are set to the same value. But you can differentiate them based on the information from your library.



Modeling Virtual Clocks

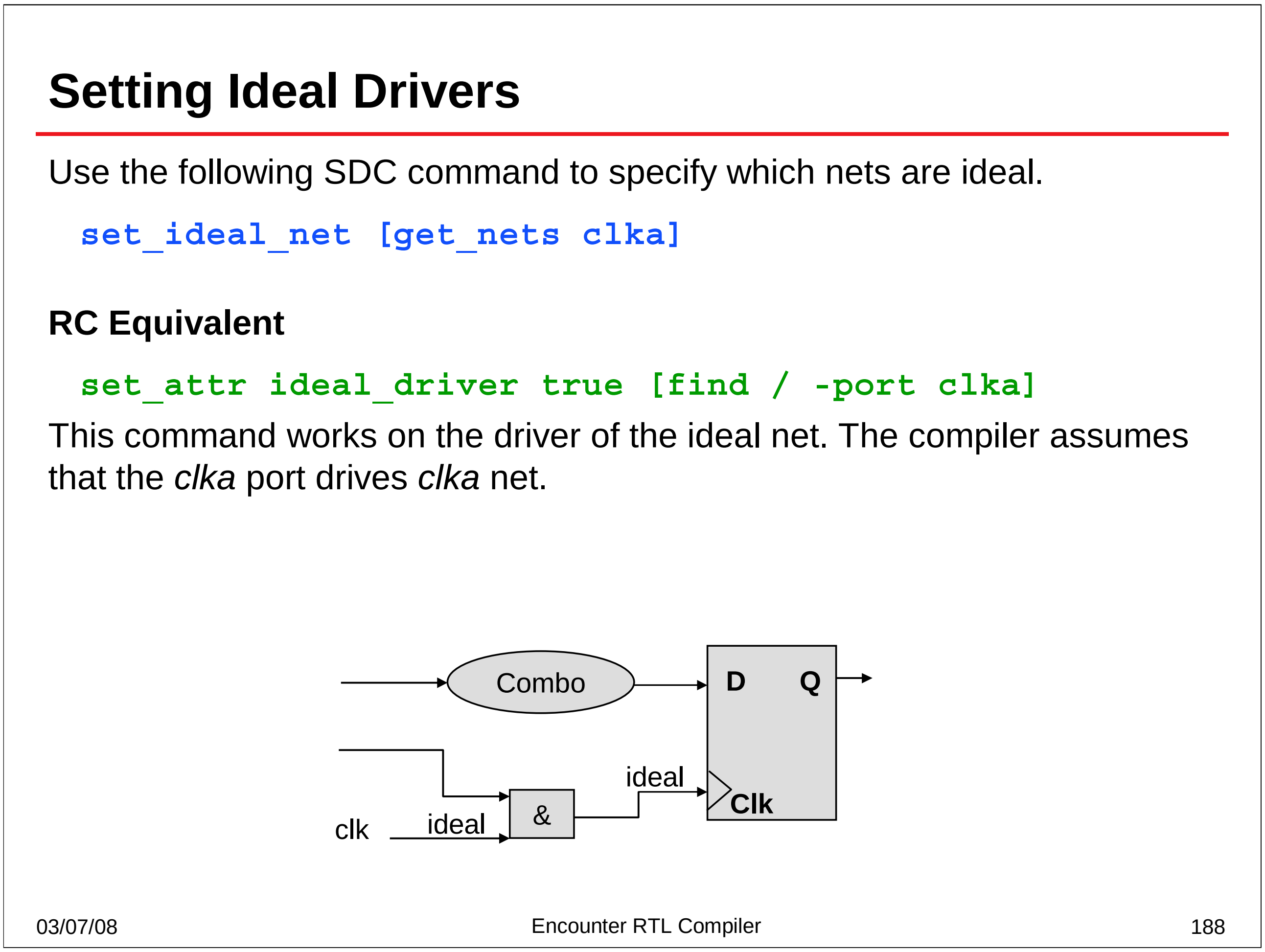
A virtual clock is a clock object that is not associated with any source in the design.

```
create_clock -period 2 -name vclock1
```

RC Equivalent

```
define_clock -period 2000 -name vclock1
```

- You can create as many clock objects as required. Use the *create_clock* command, and specify a clock name for each virtual clock.
- Use these clocks in the design to specify the timing for a combinational logic block.



Setting Ideal Drivers

Use the following SDC command to specify which nets are ideal.

```
set_ideal_net [get_nets clka]
```

RC Equivalent

```
set_attr ideal_driver true [find / -port clka]
```

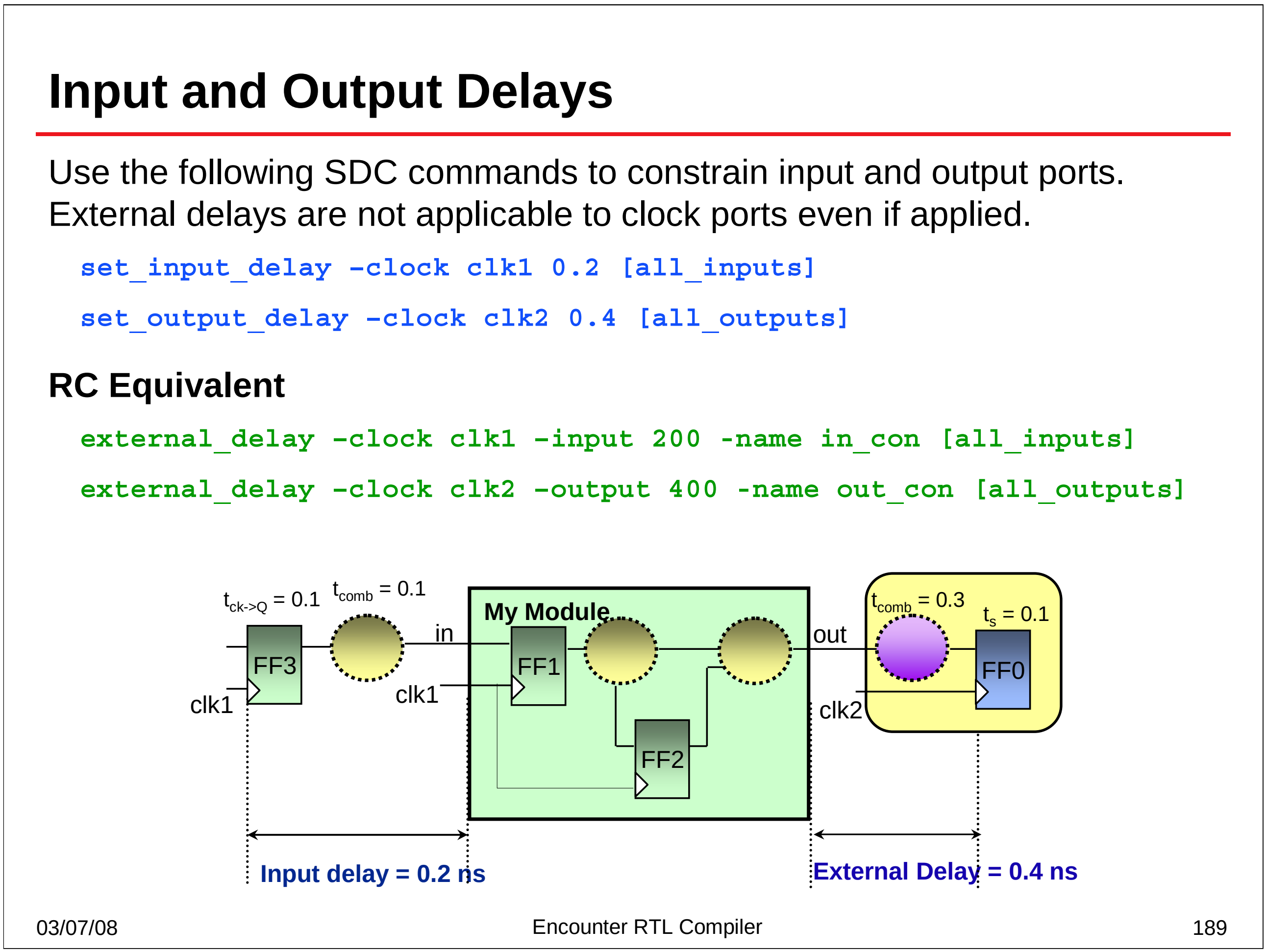
This command works on the driver of the ideal net. The compiler assumes that the *clka* port drives *clka* net.

Combo
D
Q
ideal
Clk
&
clk
ideal

03/07/08 Encounter RTL Compiler 188

When you specify a port, or a net driven by the port, is ideal the following happen:

- No DRC checks or fixes will be done on the net.
- No loading effect will be considered on the net.
- No buffer will be inserted on the net.
- No driver is upsized or split.
- Nets that drive clock pins of registers or latches through gating logic are assumed to be ideal.



Input and Output Delays

Use the following SDC commands to constrain input and output ports. External delays are not applicable to clock ports even if applied.

```
set_input_delay -clock clk1 0.2 [all_inputs]
set_output_delay -clock clk2 0.4 [all_outputs]
```

RC Equivalent

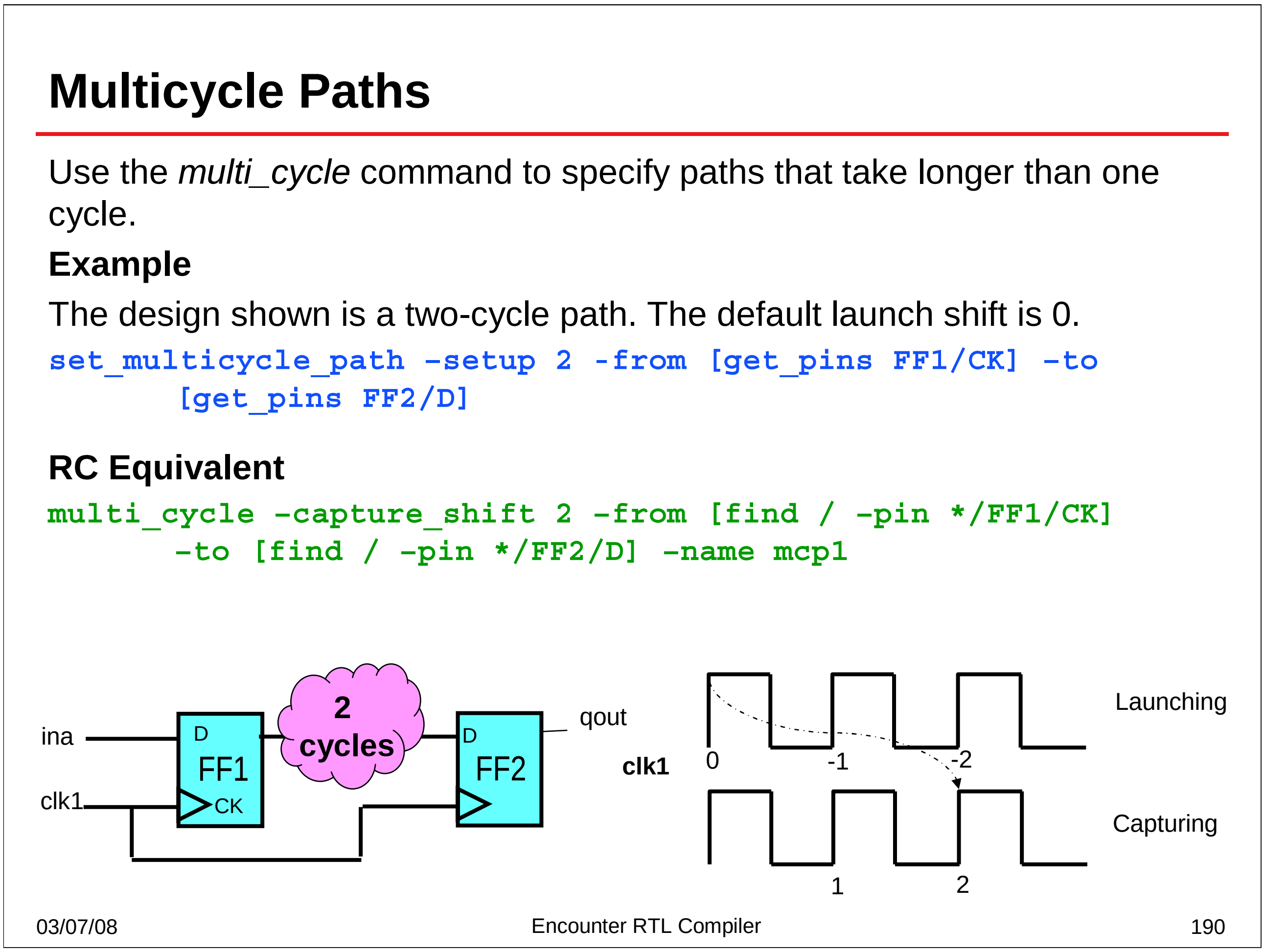
```
external_delay -clock clk1 -input 200 -name in_con [all_inputs]
external_delay -clock clk2 -output 400 -name out_con [all_outputs]
```

Timing is specified relative to the edge of a clock object as input delay and output delay. External or output delay is the delay between an output becoming stable and a capturing edge of a clock object. Input delay is the delay between a launching edge of a clock object and the time when input becomes stable.

This command sets the external port delay.

external_delay [-input number] [-output number] [-clock clock] [-edge_rise] [-edge_fall] [-level_sensitive] [-name string] (pin|port)+

- -input: (integer) clock to input valid time
- -output: (integer) output valid to clock time
- -clock: (clock) clock object
- -edge_rise: (boolean) rise clock edge
- -edge_fall: (boolean) fall clock edge
- -level_sensitive: (boolean) external event is level-sensitive
- -accumulate: (boolean) allow more than one external delay per clock per phase
- -name: (string) external delay name
- (pin|port) object(s)



Multicycle Paths

Use the *multi_cycle* command to specify paths that take longer than one cycle.

Example

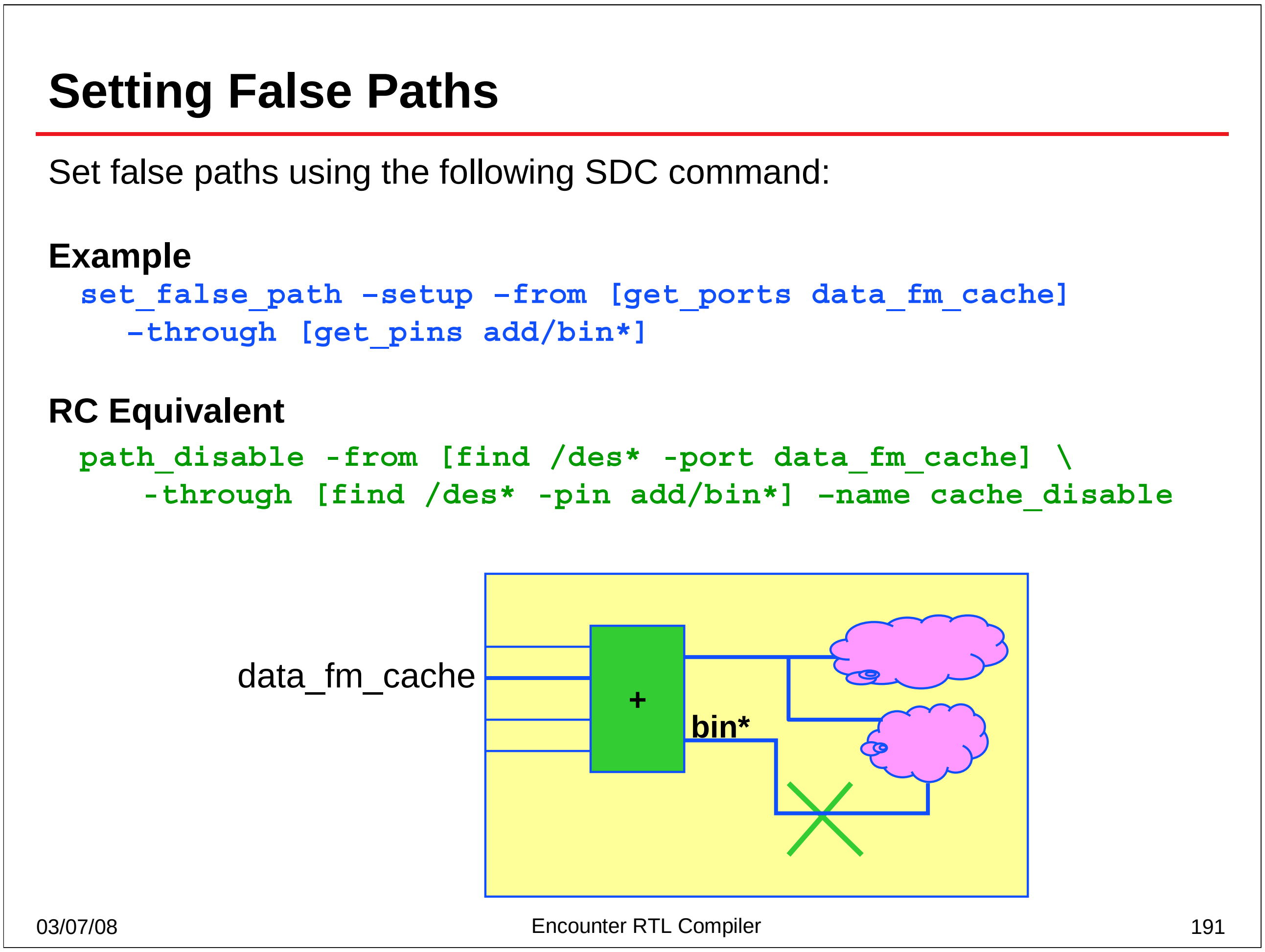
The design shown is a two-cycle path. The default launch shift is 0.

```
set_multicycle_path -setup 2 -from [get_pins FF1/CK] -to
       [get_pins FF2/D]
```

RC Equivalent

```
multi_cycle -capture_shift 2 -from [find / -pin */FF1/CK]
       -to [find / -pin */FF2/D] -name mcp1
```

The smallest positive difference between launching clock edge and capturing clock edge is the default timing constraint.



Setting False Paths

Set false paths using the following SDC command:

Example

```
set_false_path -setup -from [get_ports data_fm_cache]
  -through [get_pins add/bin*]
```

RC Equivalent

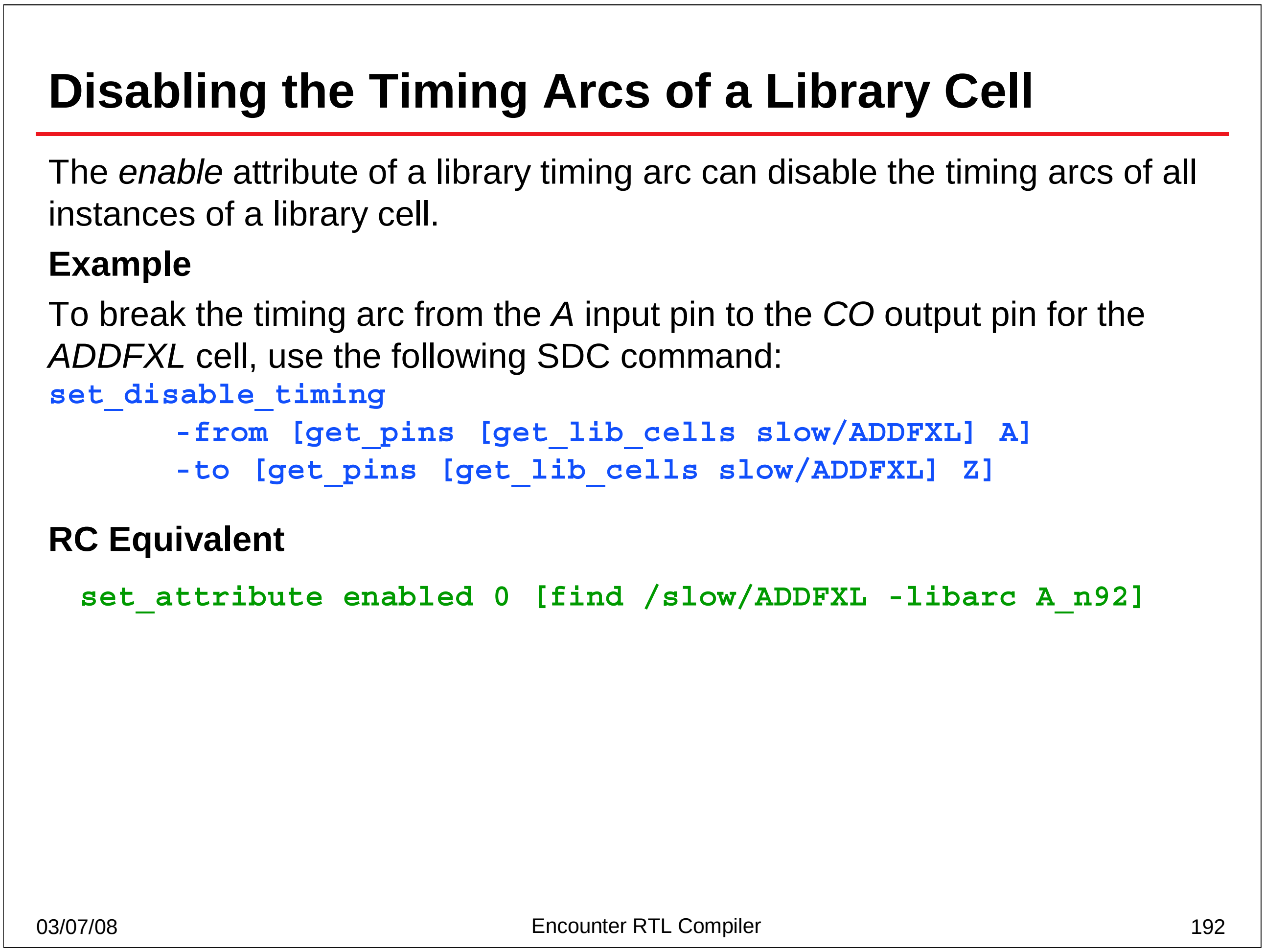
```
path_disable -from [find /des* -port data_fm_cache] \
    -through [find /des* -pin add/bin*] -name cache_disable
```

data_fm_cache
+
bin*

Set false paths on the paths that you do not want the compiler to optimize. For example, most clock-to-clock paths and paths starting from a reset or *scan_enable* pin are considered false paths.

- *from*
  Uses clock object, input port of a design, or clock pins of flip-flop.
- *to*
  Uses clock object, output ports of a design, input D pins of flip-flops.
- *through*
  Uses sequence of the points, ports, pins of hierarchical blocks or pins of sequential cells.

Encounter RTL Compiler places functional false paths between different clock domains automatically, you do not need to specify the false paths for the clocks. If you manually specify false paths between the clock domains, the ambiguity might cause RTL Compiler to synthesize the paths because it does not recognize them as false paths.



Disabling the Timing Arcs of a Library Cell

The *enable* attribute of a library timing arc can disable the timing arcs of all instances of a library cell.

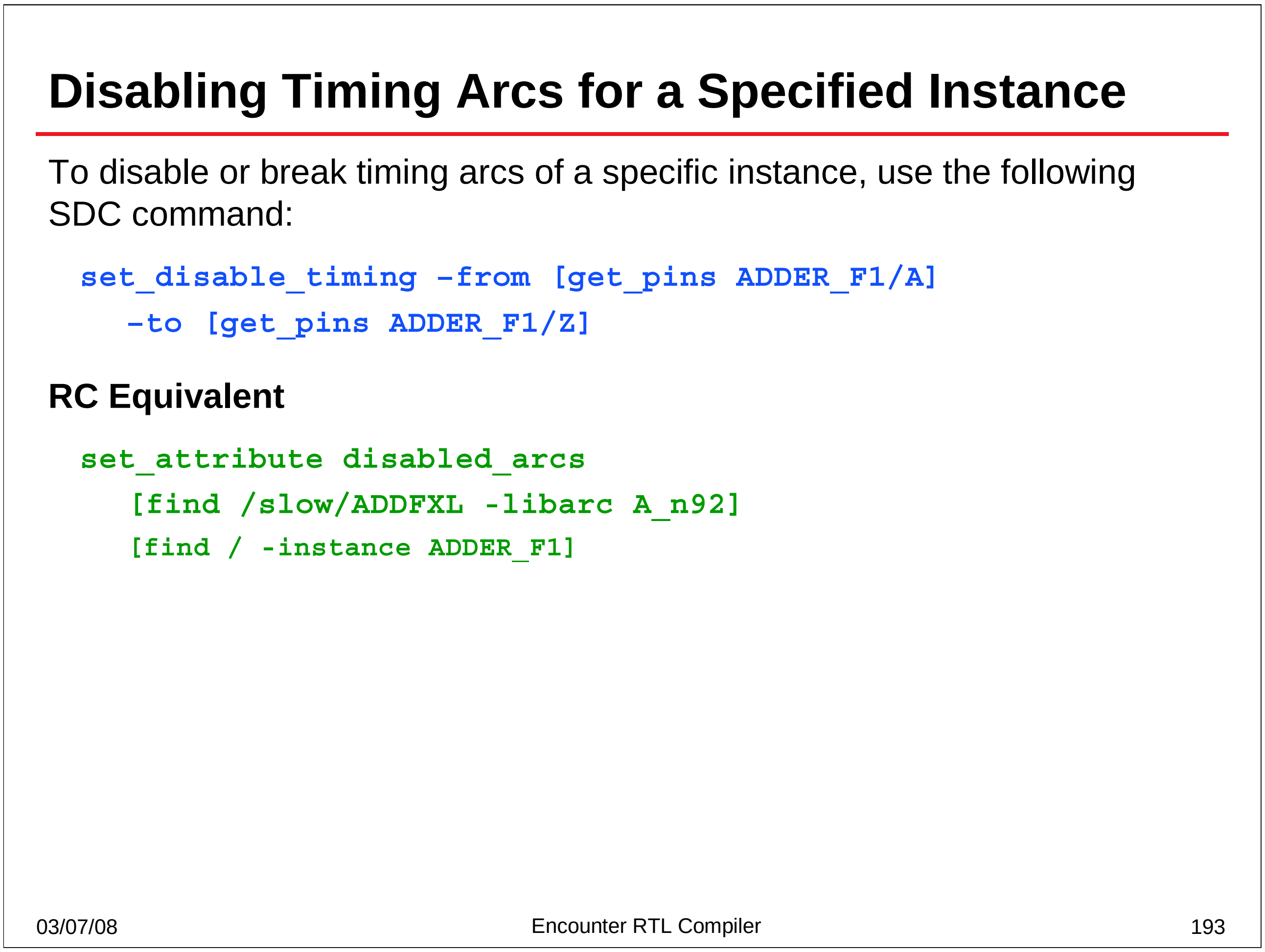
Example

To break the timing arc from the *A* input pin to the *CO* output pin for the *ADDFXL* cell, use the following SDC command:

```
set_disable_timing
       -from [get_pins [get_lib_cells slow/ADDFXL] A]
       -to [get_pins [get_lib_cells slow/ADDFXL] Z]
```

RC Equivalent

```
set_attribute enabled 0 [find /slow/ADDFXL -libarc A_n92]
```



Disabling Timing Arcs for a Specified Instance

To disable or break timing arcs of a specific instance, use the following SDC command:

```
set_disable_timing -from [get_pins ADDER_F1/A]
  -to [get_pins ADDER_F1/Z]
```

RC Equivalent

```
set_attribute disabled_arcs
   [find /slow/ADDFXL -libarc A_n92]
   [find / -instance ADDER_F1]
```

Setting Case Logic

The *set_case_analysis* command sets constant values on a port or a pin. The specified constants do not alter the netlist at the port or pin. The value set by this command has priority over the logical value from the netlist.

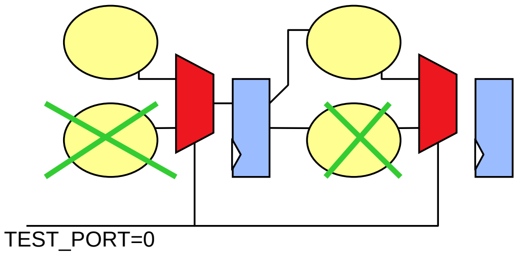
Example

```
set_case_analysis 0 [get_ports TEST_PORT]
```

RC Equivalent

```
set_attr timing_case_logic_value 0 [find / -port TEST_PORT]
```

The value for the attribute can be either a 1, 0, or a *no_value*.



You can use case analysis to prevent large transition times from propagating, whereas you cannot set a false path to prevent transition times from propagating.

A good use of case analysis is to prevent large transition times on a high fanout synchronous reset. You can also propagate the case logic downstream if loop breakers cause an interruption to optimizing the downstream logic.

You can set a constant on:

- A pin or a port to enable or disable sections of logic.
- The sequential output, but the constant cannot propagate through sequential logic.
- The input pin of a cell, but the constant does not propagate when a timing arc is disabled using *set_disable_timing*.

To get the attribute value of the case setting propagated to the MUX, use:

```
get_attr timing_case_computed_value [find / -pin <instance_name>/<pin_name>]
```

Returns 0 or 1 or no_value

To find out the *timing_case_disabled_arcs*, use:

```
get_attr timing_case_disabled_arcs [find / -instance <instance_name>]
```

Timing_case logic propagation through sequential logic can be enabled, using:

```
set_attr case_analysis_sequential_propagation <true/false>
```

Setting Path Delays

Path delays can be used to set a specific time requirement between two points. Path delays are not related to any clock, and therefore will not change if the clock frequency changes. To constrain these paths, use the *set_max_delay* SDC command.

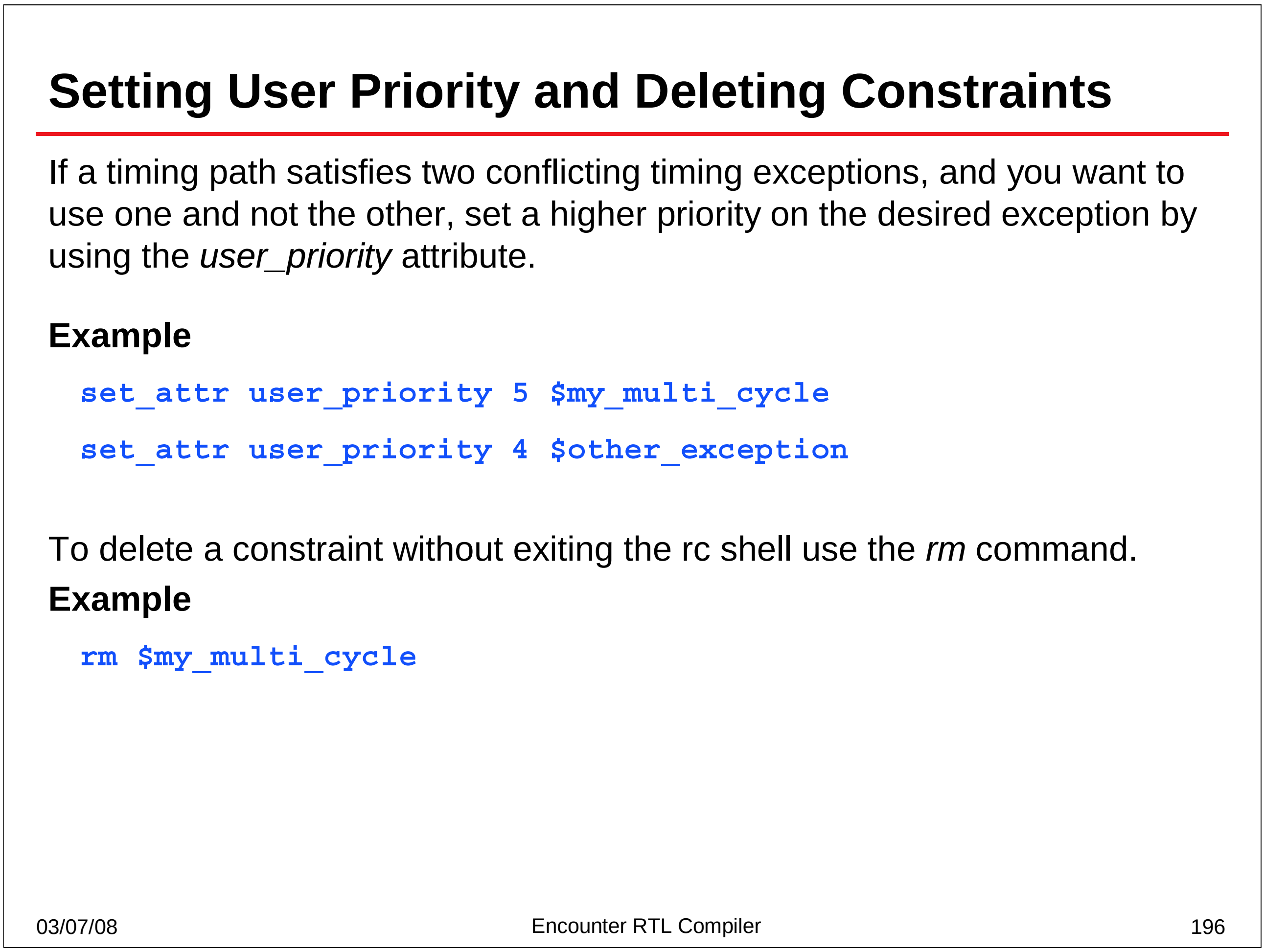
Example

```
set_max_delay 5 –from [get_ports a]
```

RC Equivalent

```
path_delay -delay 5000 -from [find / -port a] -name override
```

Setting path delays is not recommended because of the asynchronous nature of this constraint. They should be used only when absolutely necessary.



Setting User Priority and Deleting Constraints

If a timing path satisfies two conflicting timing exceptions, and you want to use one and not the other, set a higher priority on the desired exception by using the *user_priority* attribute.

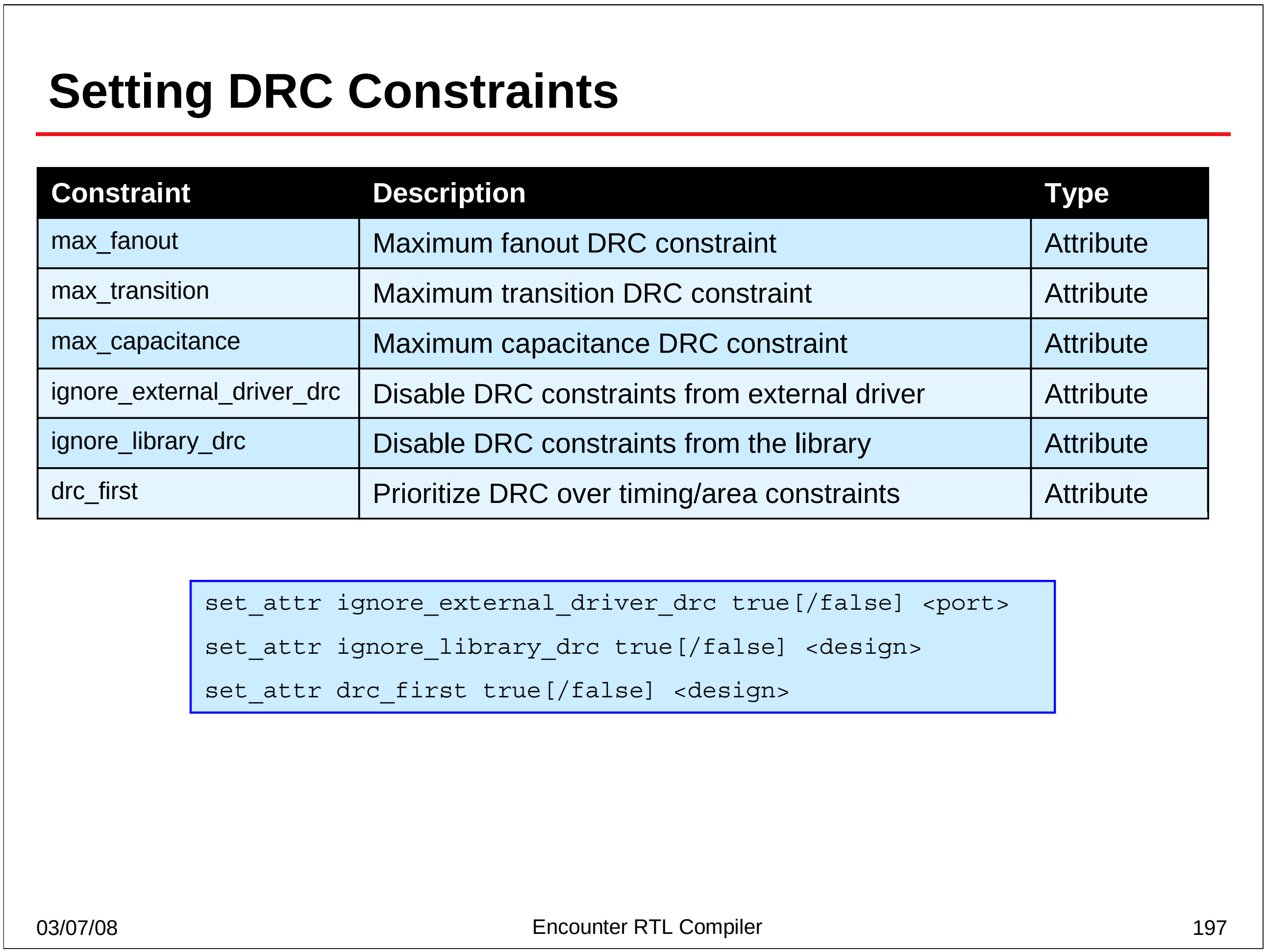
Example

```
set_attr user_priority 5 $my_multi_cycle
set_attr user_priority 4 $other_exception
```

To delete a constraint without exiting the rc shell use the *rm* command.

Example

```
rm $my_multi_cycle
```



Setting DRC Constraints

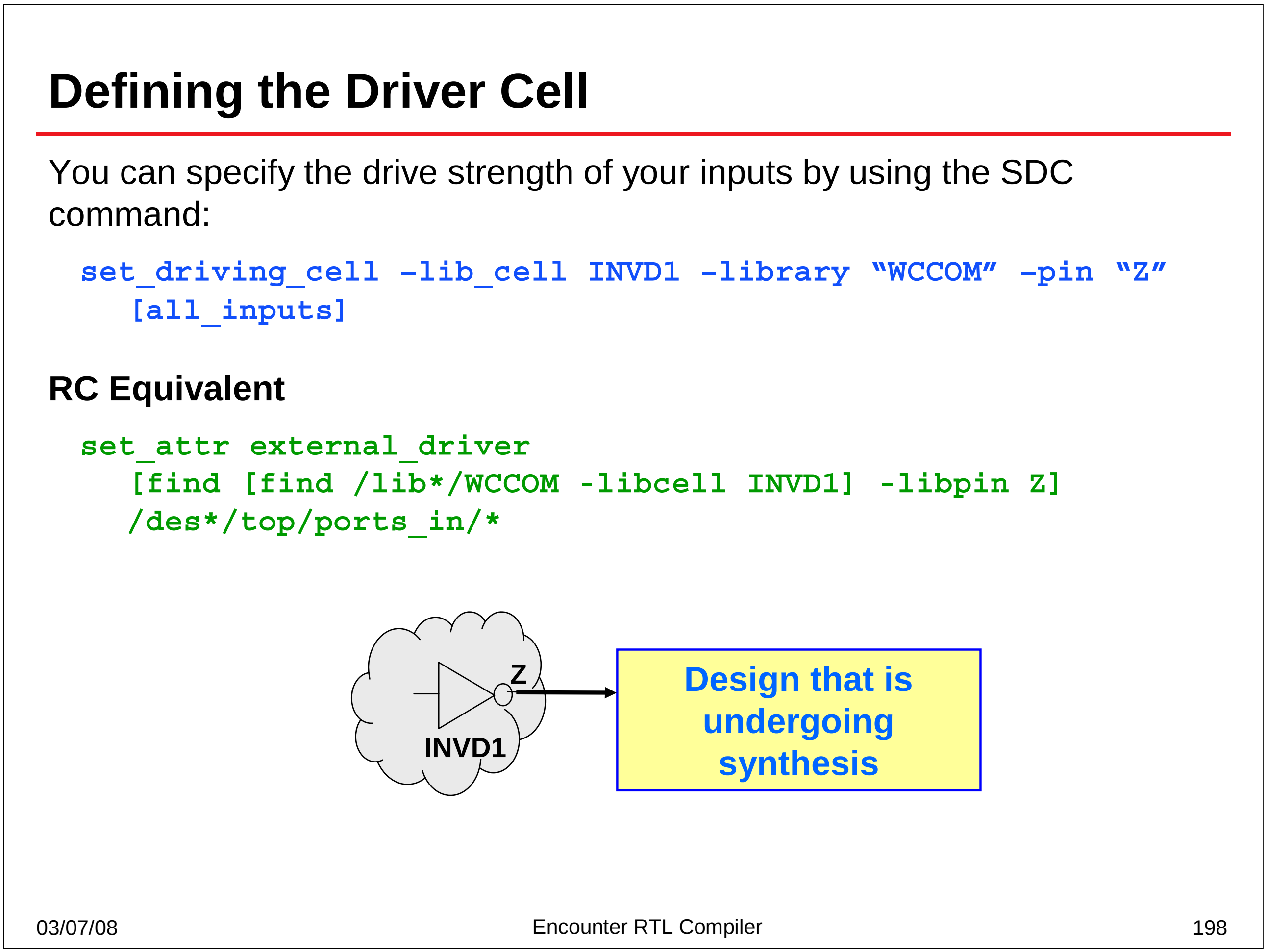
Constraint	Description	Type
max_fanout	Maximum fanout DRC constraint	Attribute
max_transition	Maximum transition DRC constraint	Attribute
max_capacitance	Maximum capacitance DRC constraint	Attribute
ignore_external_driver_drc	Disable DRC constraints from external driver	Attribute
ignore_library_drc	Disable DRC constraints from the library	Attribute
drc_first	Prioritize DRC over timing/area constraints	Attribute

```
set_attr ignore_external_driver_drc true[/false] <port>
set_attr ignore_library_drc true[/false] <design>
set_attr drc_first true[/false] <design>
```

03/07/08 Encounter RTL Compiler 197

Here are some examples of DRC constraints and their SDC equivalent commands:

```
set_attr max_fanout 2 [find / -port out*]
dc::set_max_fanout 2 [dc::get_ports out*]

set_attr max_capacitance 100
dc::set_max_capacitance 0.1

set_attr max_transition 80
dc::set_max_transition 0.08
```



Defining the Driver Cell

You can specify the drive strength of your inputs by using the SDC command:

```
set_driving_cell -lib_cell INVD1 -library “WCCOM” -pin “Z”
   [all_inputs]
```

RC Equivalent

```
set_attr external_driver
   [find [find /lib*/WCCOM -libcell INVD1] -libpin Z]
   /des*/top/ports_in/*
```

Design that is undergoing synthesis

You can also specify the input resistance instead of the input driving cell.

Steps to Define Input Driver Strength

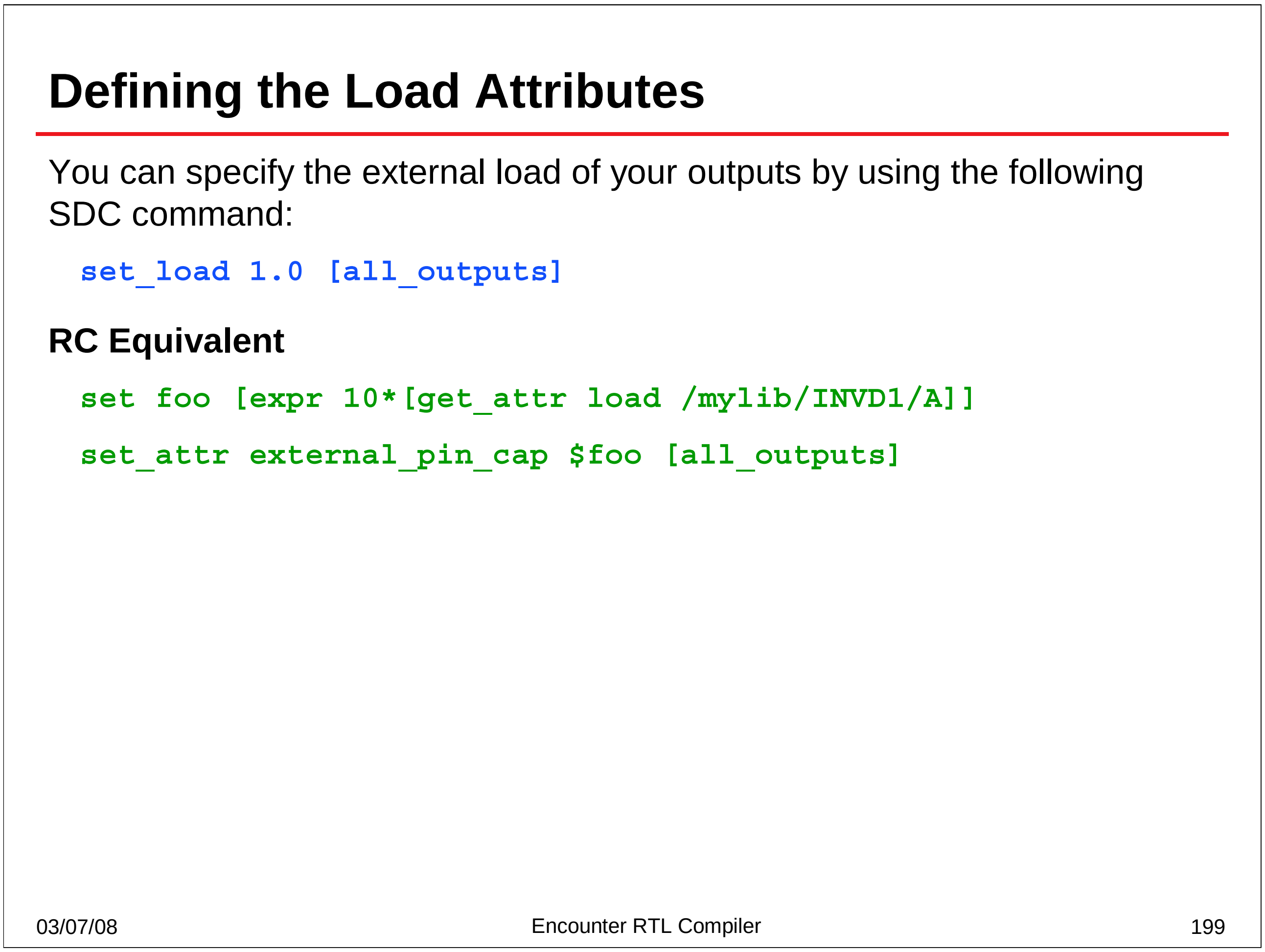
- Find the name of the library cell driving the port.
- Find the driver pin of the library cell.
- Assign the driver cell information to the input ports.

The equivalent syntax for setting *external_driver* attribute on all input pins is

```
set_attr external_driver /lib*/mylib/libcells/INVD1/Z [all_inputs]
```

To remove the driving cell from clock pins, use

```
set_attribute external_driver "" [find / -port CK_TD]
```



Defining the Load Attributes

You can specify the external load of your outputs by using the following SDC command:

```
set_load 1.0 [all_outputs]
```

RC Equivalent

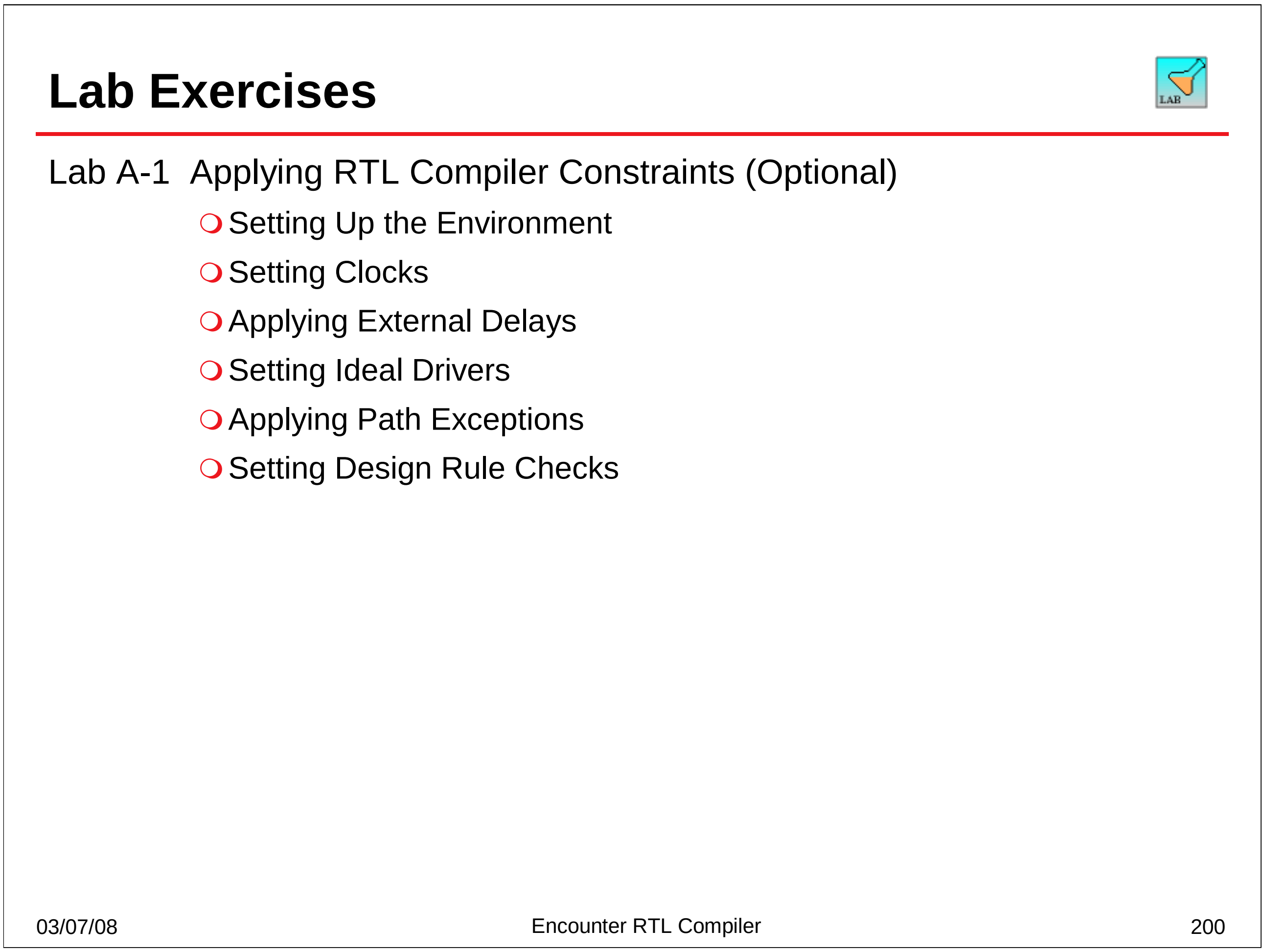
```
set foo [expr 10*[get_attr load /mylib/INVD1/A]]
set_attr external_pin_cap $foo [all_outputs]
```

03/07/08 Encounter RTL Compiler 199

You can also define output load per fanout and port fanout values instead of specifying the external load.

Steps

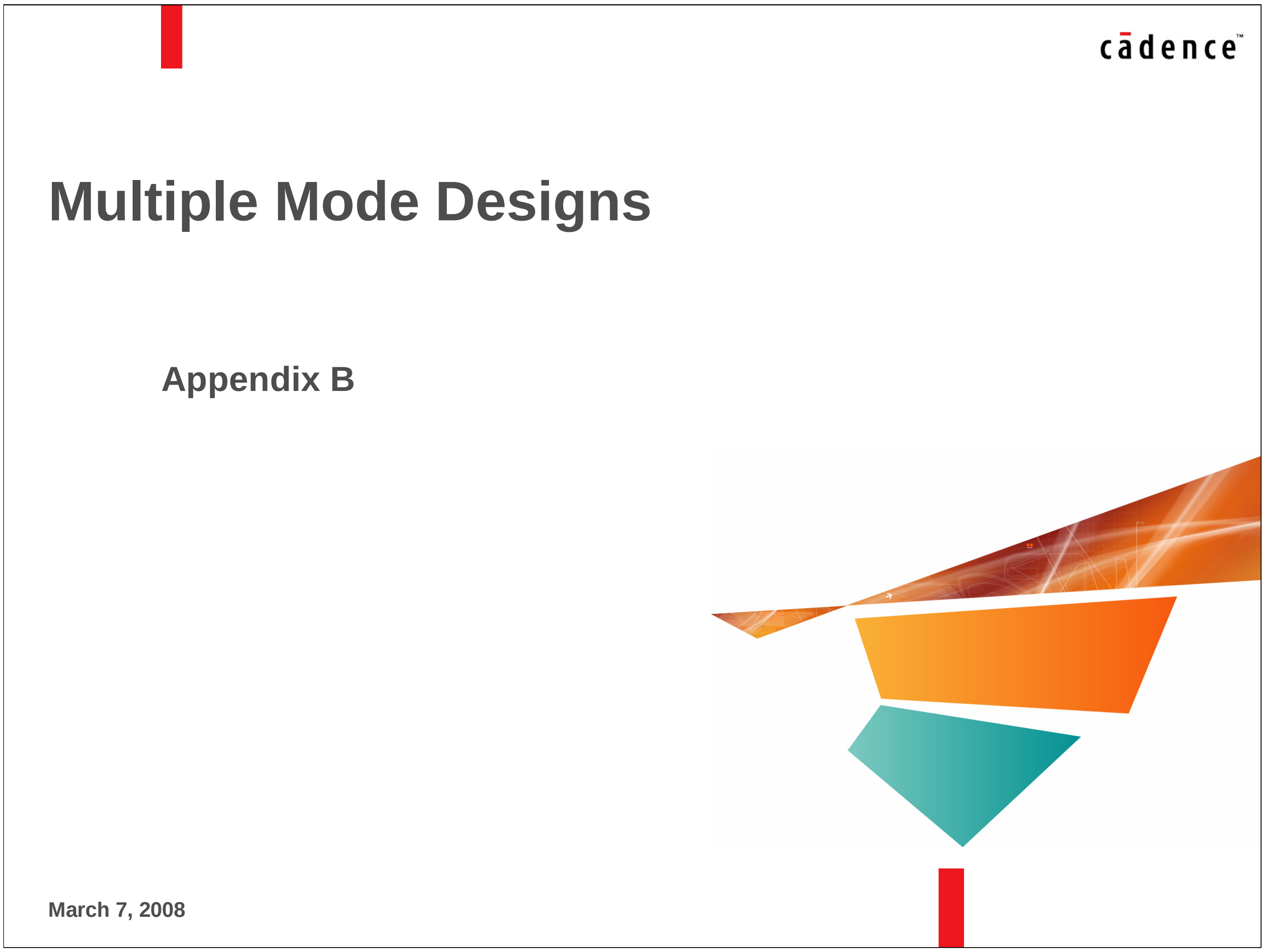
- Get the value (load attribute) from the load cell (for example INVD1).
- Use *external_pin_cap* attribute to set the capacitance load on output ports that drive this load cell.



Lab Exercises

Lab A-1 Applying RTL Compiler Constraints (Optional)

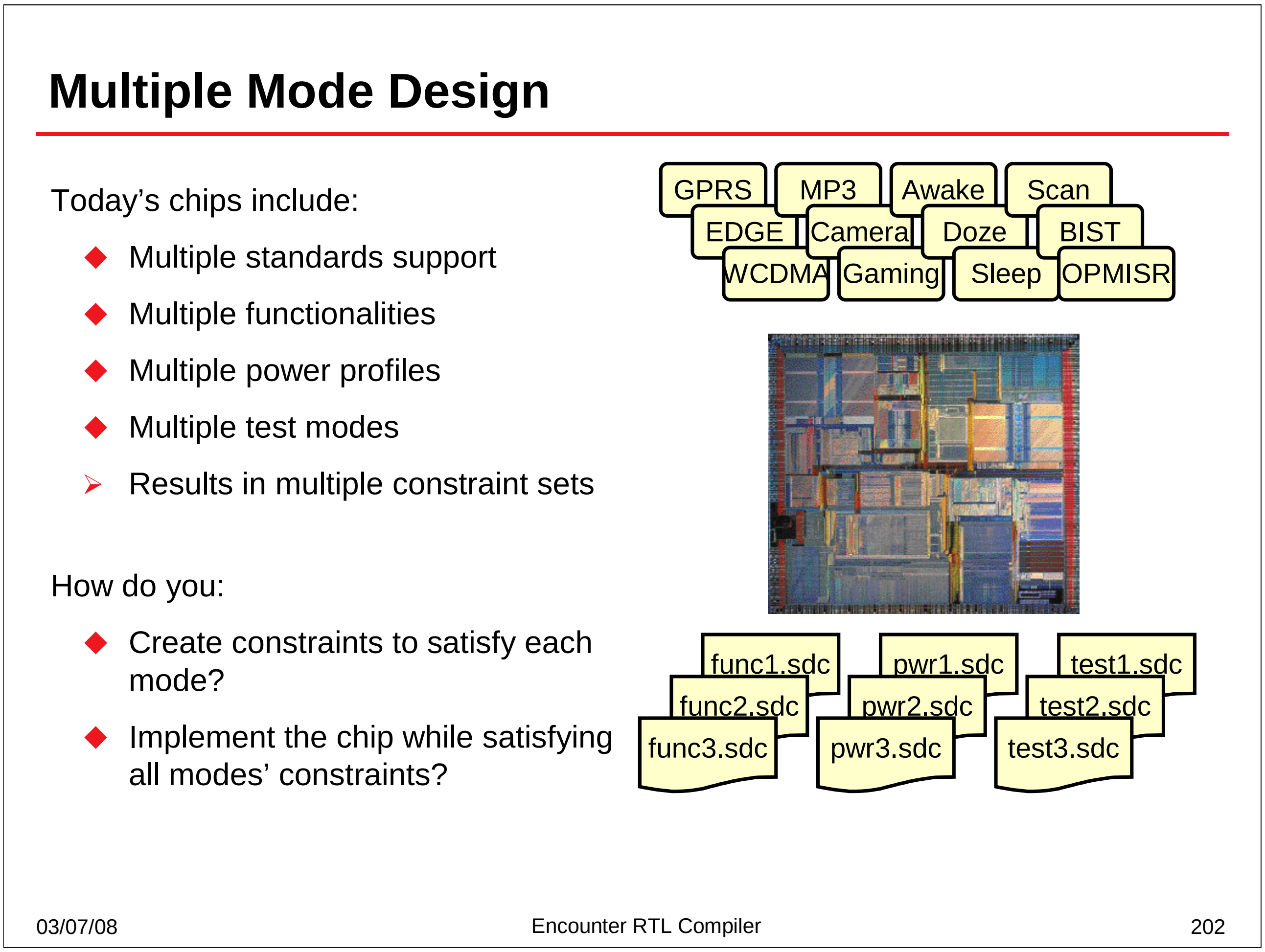
- Setting Up the Environment
- Setting Clocks
- Applying External Delays
- Setting Ideal Drivers
- Applying Path Exceptions
- Setting Design Rule Checks



Multiple Mode Designs

Appendix B

March 7, 2008



Multiple Mode Design

Today’s chips include:

- Multiple standards support
- Multiple functionalities
- Multiple power profiles
- Multiple test modes
- Results in multiple constraint sets

How do you:

- Create constraints to satisfy each mode?
- Implement the chip while satisfying all modes’ constraints?

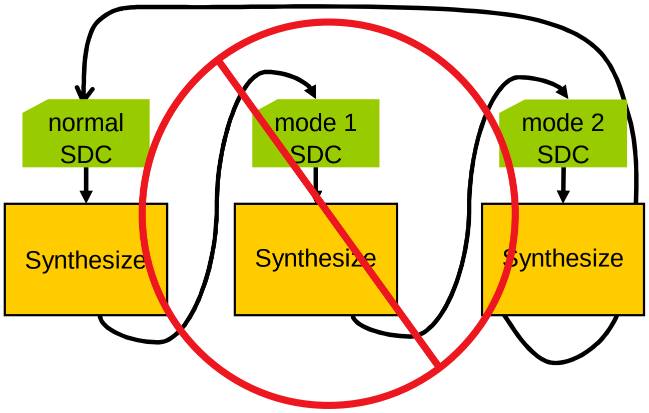
GPRS
MP3
Awake
Scan
EDGE
Camera
Doze
BIST
WCDMA
Gaming
Sleep
OPMISR
func1.sdc
func2.sdc
func3.sdc
pwr1.sdc
pwr2.sdc
pwr3.sdc
test1.sdc
test2.sdc
test3.sdc

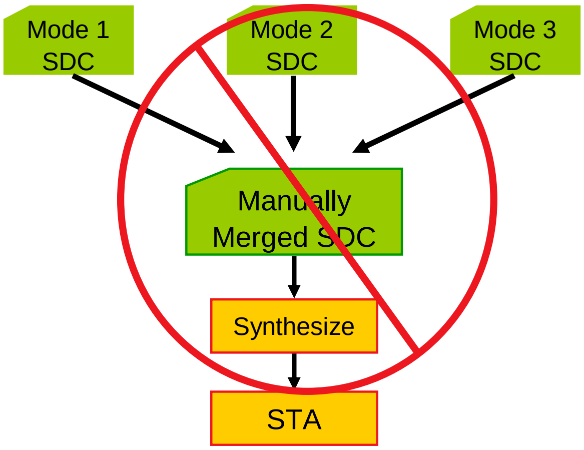
Chip design today is becoming very complicated due to integration. We constantly see announcements touting single-chip solutions that support multiple standards, or that perform multiple functions. In addition to today’s power densities, chips operate in different modes to conserve power. And where there used to be only one test mode, now there can be many.

This means multiple sets of constraints, such as clocks, external delays, false paths, and multicycle paths. How can we satisfy all these constraints while still meeting today’s time-to-market demands?

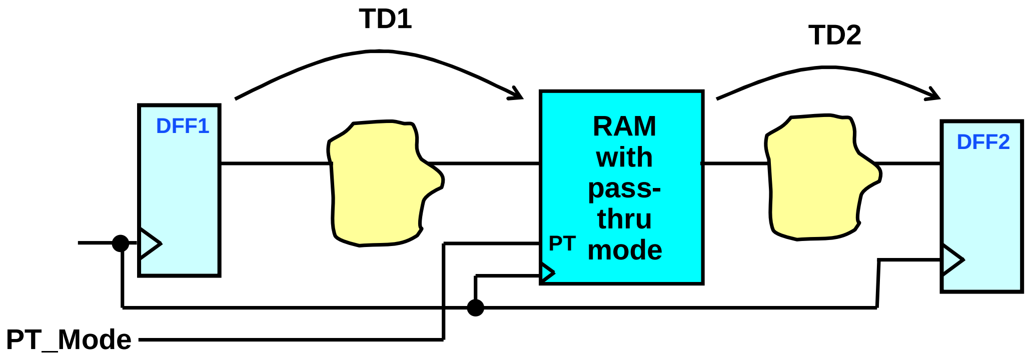
Multi-Mode: Traditional Flows

- Serial timing optimization collapses the previous optimization result. You need to optimize all modes concurrently.
- You can merge the SDCs manually, but it is error prone.





Example of Multi-Mode



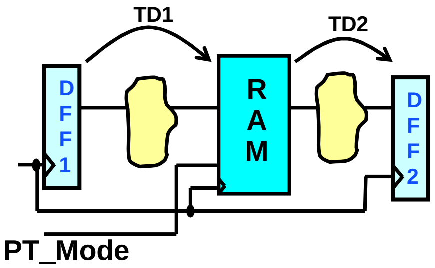
Mode 1: Non-bypass mode timing paths

- Path 1: $\text{DFF1}_{CK\rightarrow Q}$ + TD1 – RAM $_{\text{setup}}$
- Path 2: $\text{RAM}_{D\rightarrow Q}$ + TD2 – $\text{DFF2}_{\text{setup}}$

Mode 2: Bypass mode timing paths

- $\text{DFF1}_{CK\rightarrow Q}$ + TD1 + $\text{RAM}_{D\rightarrow Q}$ + TD2 – $\text{DFF2}_{\text{setup}}$

Example of Multi-Mode (continued)



$DFF1_{CK \rightarrow Q}$: 160ps
RAM $_{setup}$: 200ps
$RAM_{D \rightarrow Q}$: 300ps
$DFF2_{setup}$: 160ps

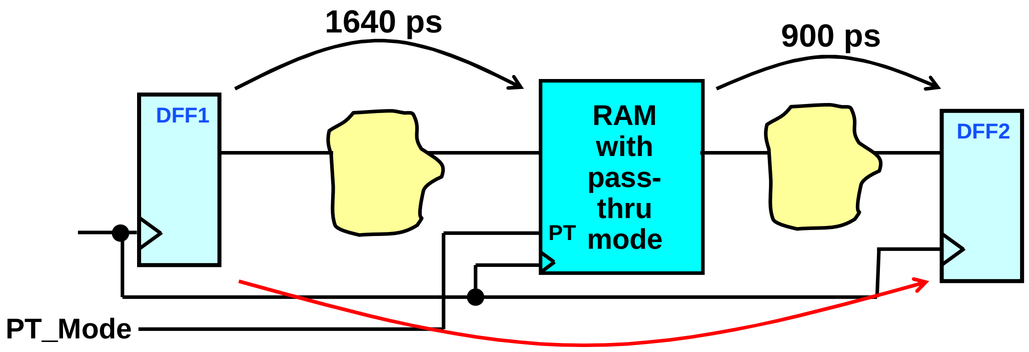
Non-bypass @ CP1 = 500 MHz

- TD1/TD2 requirement: CP – $DFF1_{CK \rightarrow Q}$ – RAM $_{setup}$
  - 2000 – 160 – 200 = 1680 ps

Bypass @ CP2 = 300 MHz

- DFF1 → DFF2 requirement: CP2 – $DFF1_{CK \rightarrow Q}$ – $DFF2_{setup}$ – $RAM_{D \rightarrow Q}$
  - 3333 – 160 – 160 – 300 = 2713 ps

Example of Multi-Mode: Solution 1



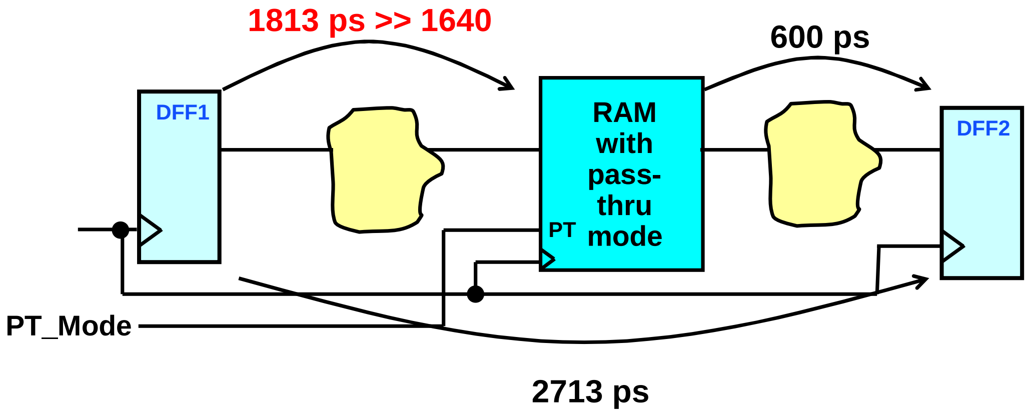
1640 + 300 + 900 = 2840 (>2713)

Step 1: Synthesize with non-bypass (faster) constraints.

Step 2: Analyze with bypass (slower) constraints.

Step 3: Resynthesize with bypass constraints, while preserving all other logic in the ENTIRE design.

Example of Multi-Mode: Solution 2



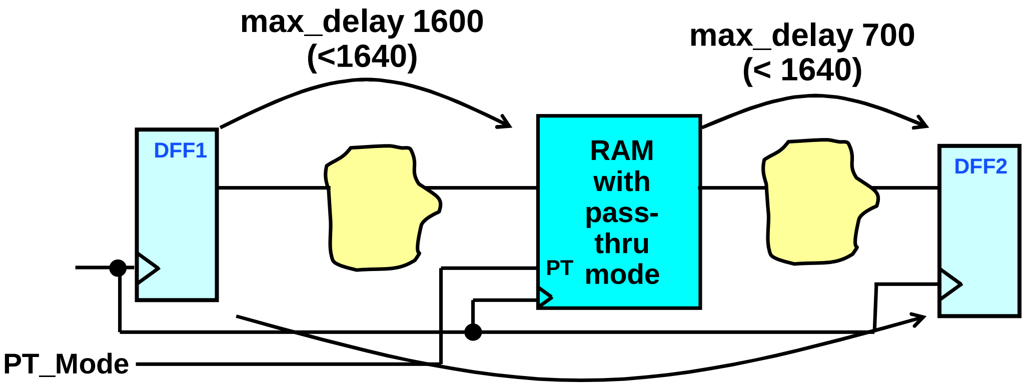
Step 1: Synthesize with bypass constraints.

Step 2: Analyze with non-bypass constraints.

Step 3: Resynthesize with non-bypass constraints, while preserving all other logic in the ENTIRE design.

Example of Multi-Mode: Solution 3

Manually Create a Time Budget



max_delay 1600
(<1640)

max_delay 700
(< 1640)

DFF1

RAM
with
pass-
thru
mode

PT

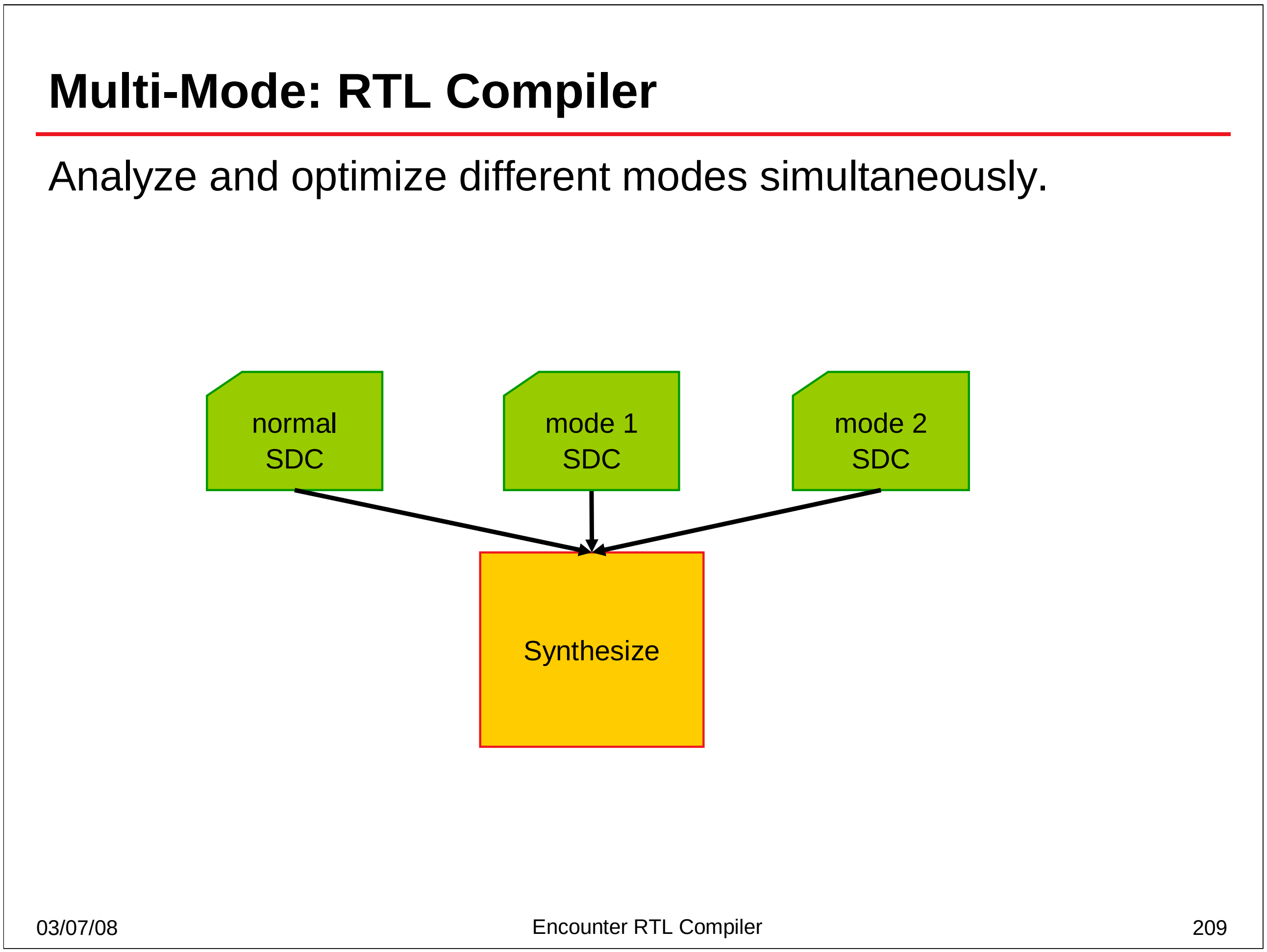
DFF2

PT_Mode

1600 + 300 + 700 = 2600 (< 2713)

It looks good, but you are in trouble if…

- Your manager says that the bypass mode needs to run at 350 MHZ.
- The output logic of the RAM has been changed.



Multi-Mode: RTL Compiler

Analyze and optimize different modes simultaneously.

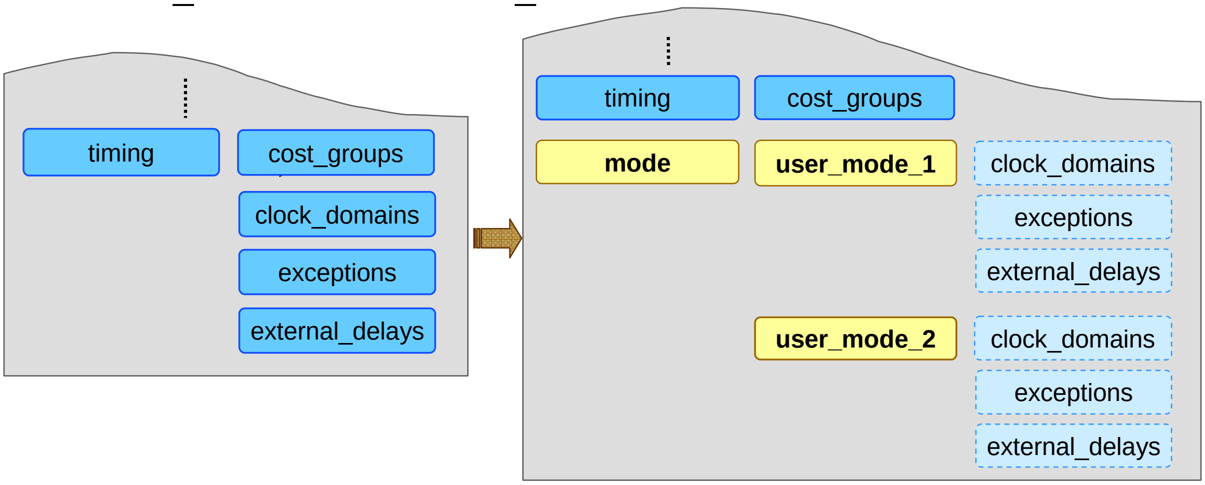
Multi-Mode Commands

- Set up the multi-mode analysis with this command:

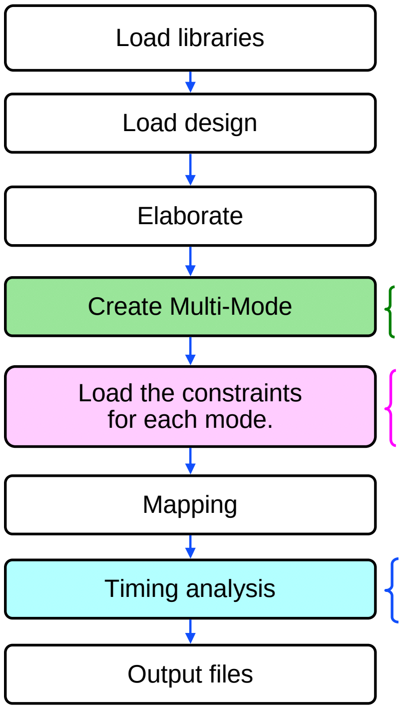
```
create_mode -name <list_of_modes>
```

- Annotate each mode with the corresponding SDC files with this command:

```
read_sdc -mode <mode_name> constraints.sdc
```



Multi-Mode Flow



```
create_mode -name “mode1 mode2” [-design <design_name>]
```

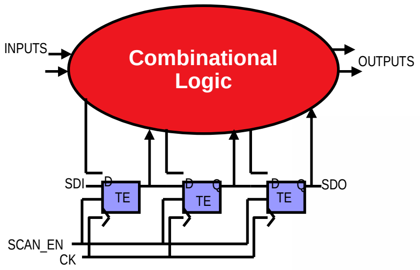
```
read_sdc -mode mode1 <mbist_1.sdc> <function_1.sdc>
read_sdc -mode mode2 <mbist_2.sdc> <function_2.sdc>
```

```
report timing
report timing -mode [find / -mode mode1]
```

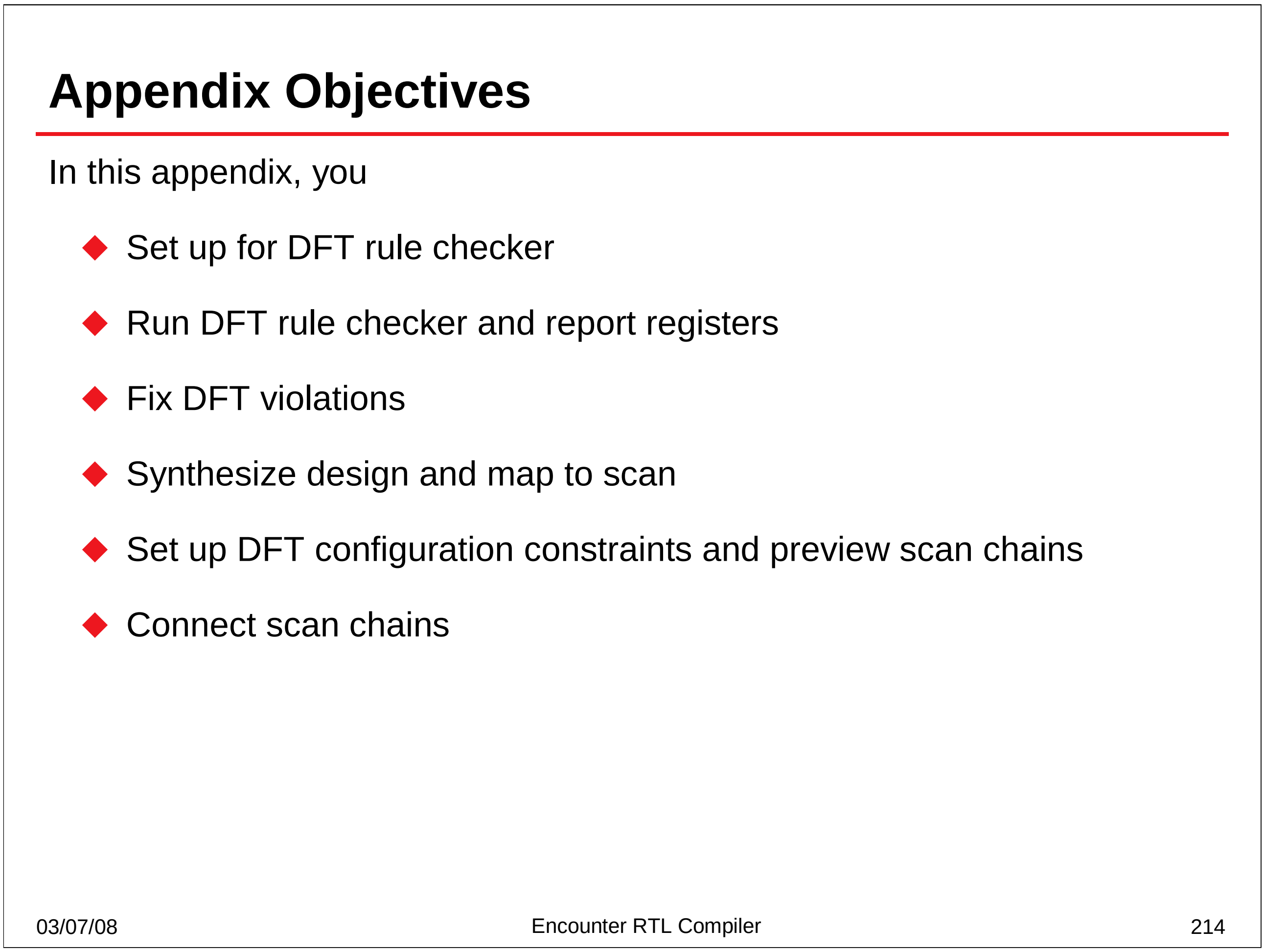
If you are using the multi-mode flow, then you must specify the *-mode* option to each of the commands.

Test Synthesis

Appendix C



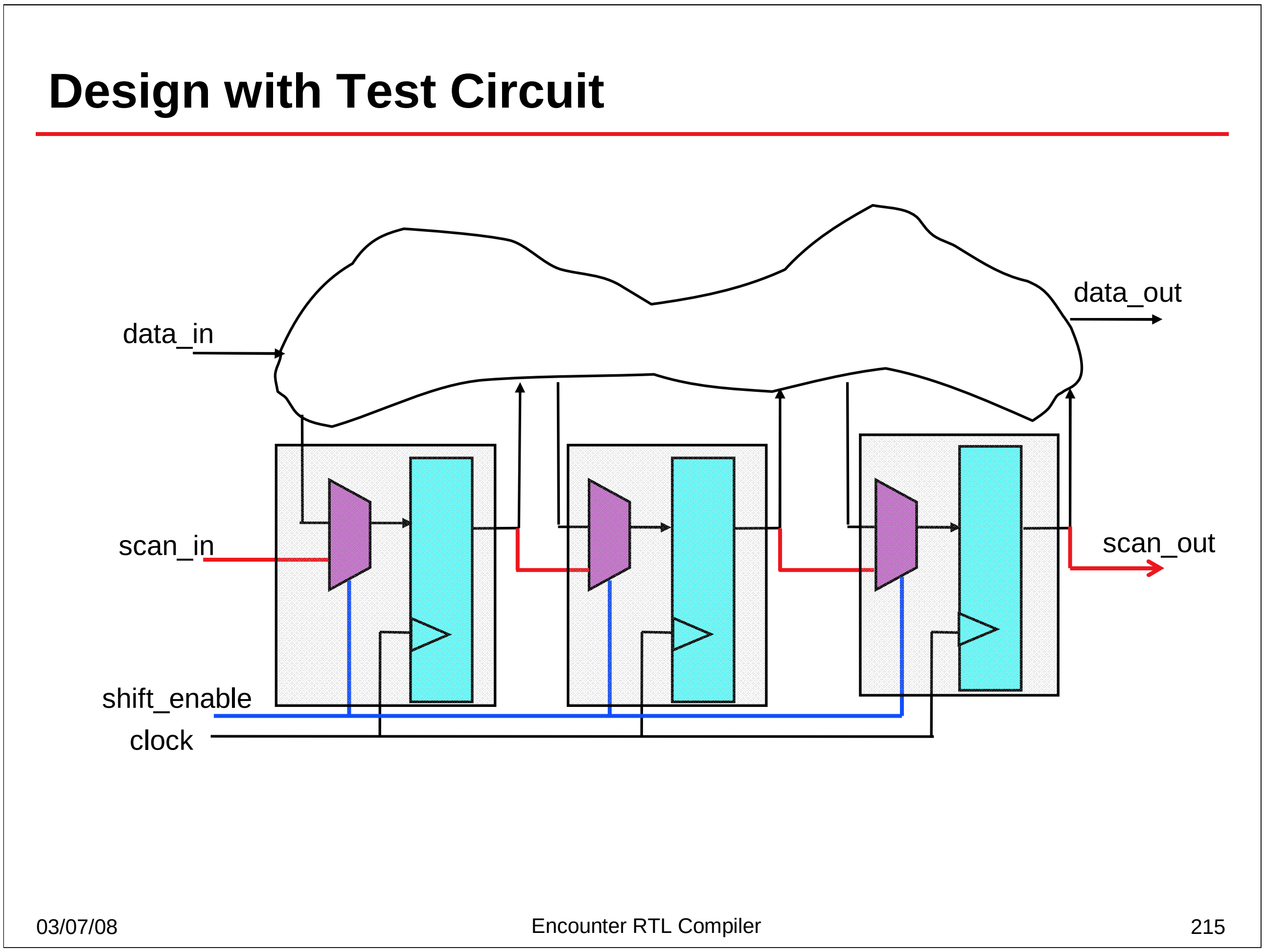
March 7, 2008



Appendix Objectives

In this appendix, you

- Set up for DFT rule checker
- Run DFT rule checker and report registers
- Fix DFT violations
- Synthesize design and map to scan
- Set up DFT configuration constraints and preview scan chains
- Connect scan chains



Design with Test Circuit

03/07/08 Encounter RTL Compiler 215

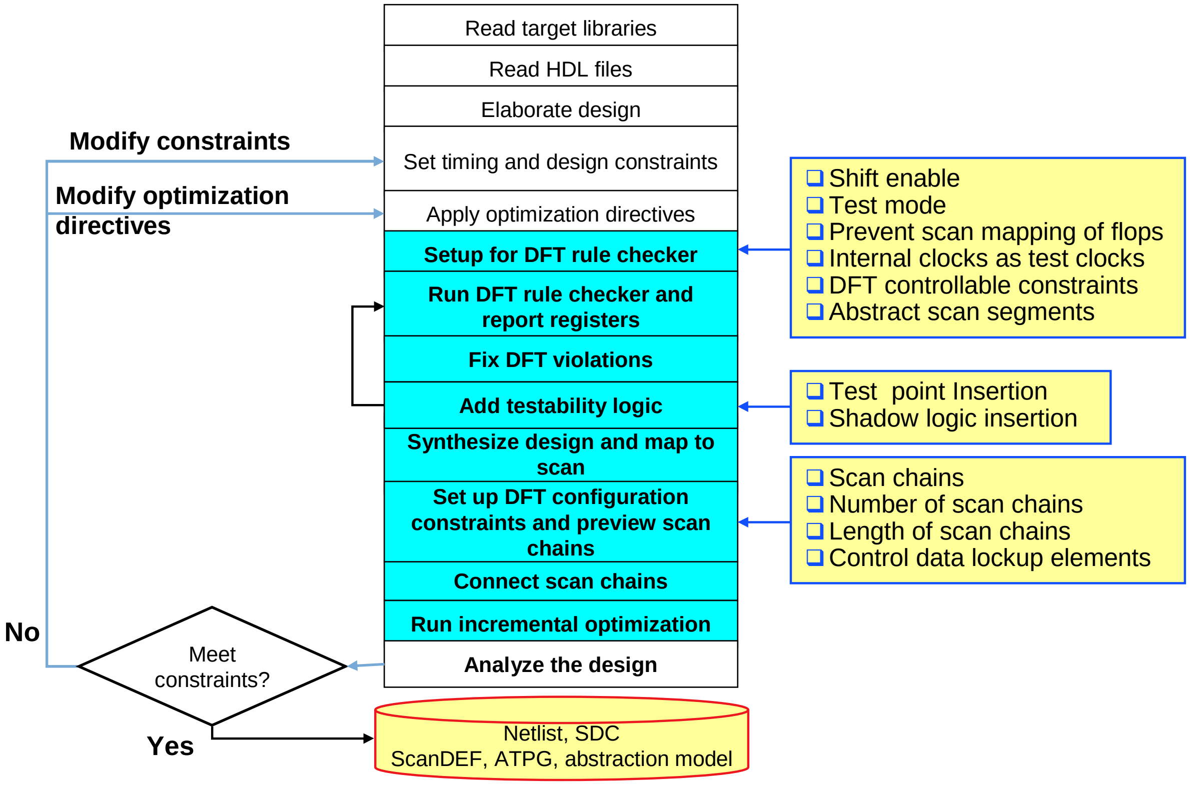
Any inferred register, or any instantiated edge triggered registers that pass the DFT rule checks will be mapped to their scan equivalent cell when the scan connection engine runs.

Remapping of instantiated edge-triggered registers that pass the DFT rule checks to their scan equivalent cells, occurs prior to placement only.

The test process refers to testing the ASIC for manufacturing defects on the automatic test equipment (ATE). It is the process of analyzing the logic on the chip to detect logic faults. The test process puts the circuit into one of the three test modes:

- Capture mode
  This is the part of the test process that analyzes the combinational logic on the chip. The registers act first as pseudo-primary inputs (using ATPG-generated test data), and then as pseudo-primary outputs (capturing the output of the combinational logic).
- Scan-shift mode
  This is the part of the test process in which registers act as shift registers in a scan chain. Test vector data is shifted into the scan chain registers, and the captured data from capture mode, are shifted out of the scan registers.
- System mode
  This is the normal or intended operation of the circuit. Any logic dedicated for DFT purposes is NOT active in system mode.

RTL Top-down Design-for-Testability (DFT) Flow



Read target libraries

Read HDL files

Elaborate design

Set timing and design constraints

Apply optimization directives

Setup for DFT rule checker

Run DFT rule checker and report registers

Fix DFT violations

Add testability logic

Synthesize design and map to scan

Set up DFT configuration constraints and preview scan chains

Connect scan chains

Run incremental optimization

Analyze the design

Meet constraints?

No

Modify constraints

Modify optimization directives

Yes

Netlist, SDC
ScanDEF, ATPG, abstraction model

- ❑ Shift enable
- ❑ Test mode
- ❑ Prevent scan mapping of flops
- ❑ Internal clocks as test clocks
- ❑ DFT controllable constraints
- ❑ Abstract scan segments

- ❑ Test point Insertion
- ❑ Shadow logic insertion

- ❑ Scan chains
- ❑ Number of scan chains
- ❑ Length of scan chains
- ❑ Control data lockup elements

03/07/08 Encounter RTL Compiler 216

Encounter® RTL Compiler provides a full-scan DFT solution including:

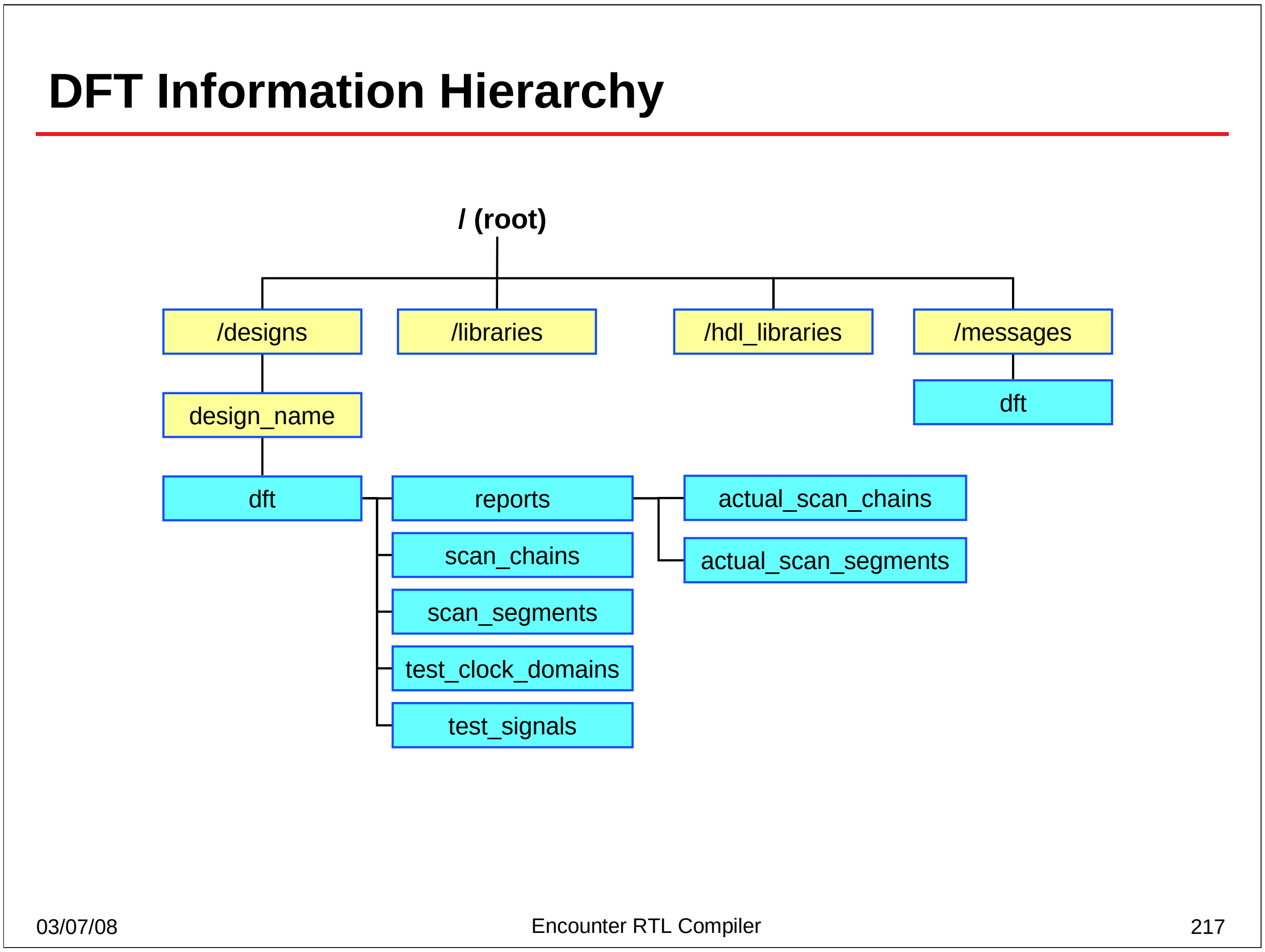
- DFT rule checking
- DFT rule violation fixing
- Scan mapping during optimization
- Scan chain configuration and connection

The main DFT techniques available today are:

- Scan insertion
- Memory BIST insertion
- Logic BIST insertion
- Boundary scan insertion

Scan insertion is one of the most used DFT techniques to detect stuck-at-faults.

Scan insertion replaces the flip-flops in the design with special flops that contain built-in logic targeted for testability. Scan logic lets you control and observe the *sequential state* of the design through the test pins during test mode. This helps in generating a high quality and compact test pattern set for the design using an Automatic Test Pattern Generator (ATPG) tool.



DFT Information Hierarchy

- / (root)
  - /designs
    - design_name
      - dft
        - reports
          - actual_scan_chains
          - actual_scan_segments
        - scan_chains
        - scan_segments
        - test_clock_domains
        - test_signals
  - /libraries
  - /hdl_libraries
  - /messages
    - dft

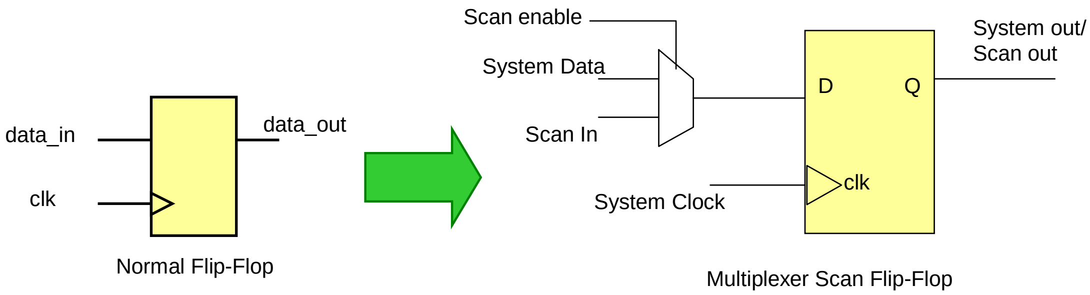
Scan Styles: MUX Scan

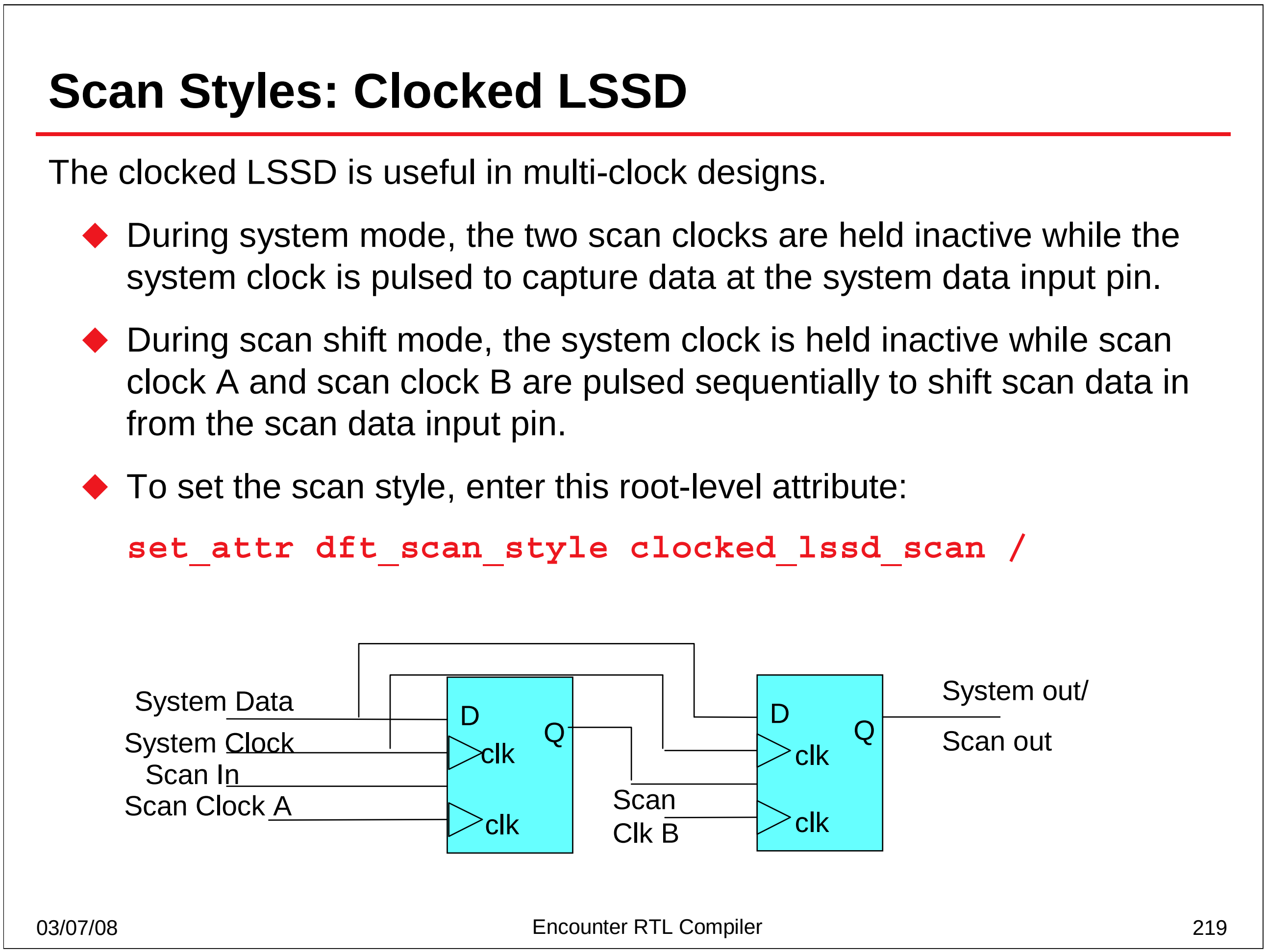
The muxed scan style (*muxed_scan*) is the most commonly used scan style.

- Only one clock is used for both system and scan mode.
- The enable signal drives either the scan data or the system data.

To set the scan style, enter the following root-level attribute:

```
set_attr dft_scan_style muxed_scan /
```





Scan Styles: Clocked LSSD

The clocked LSSD is useful in multi-clock designs.

- During system mode, the two scan clocks are held inactive while the system clock is pulsed to capture data at the system data input pin.
- During scan shift mode, the system clock is held inactive while scan clock A and scan clock B are pulsed sequentially to shift scan data in from the scan data input pin.
- To set the scan style, enter this root-level attribute:

```
set_attr dft_scan_style clocked_lssd_scan /
```

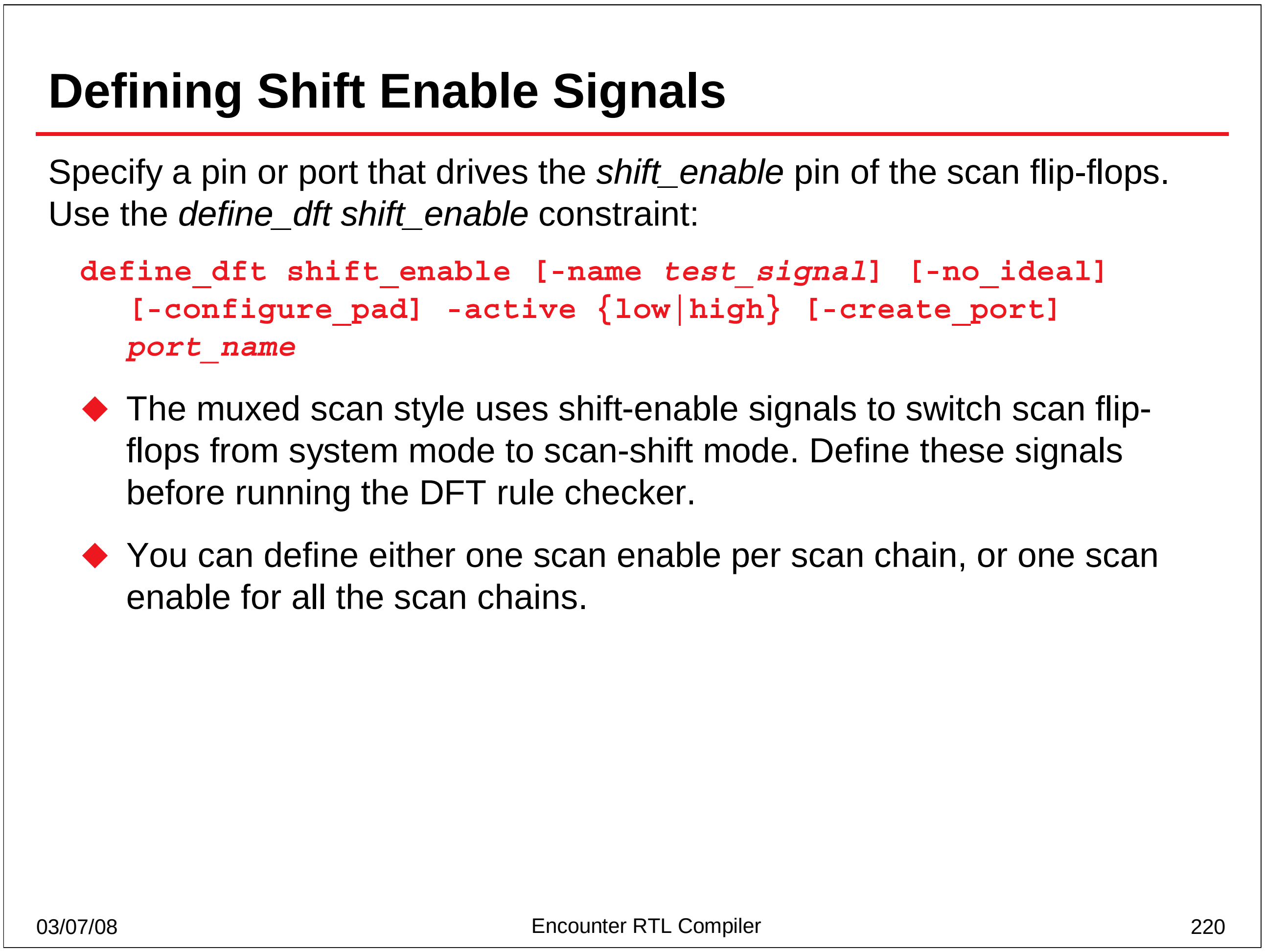
In the clocked-LSSD scan style, an edge-triggered D-flop is replaced by a clocked-LSSD scan cell that has one edge triggered on system clock and two level-sensitive scan clocks: scan clock A and scan clock B.

Pros

- Minimal performance hit.
- Suitable for latch-based designs, although test synthesis only supports an LSSD variant of an edge triggered D-flop.
- Relaxed DFT rule checks.
- Can capture and shift out the state of the scan flip flop with out affecting the functional state of the circuit.
- Can debug/diagnose a problem while operating the design in system mode.
- Easier for ATPG to get higher fault coverage in partial scan designs.

Cons

- Much more complex cell and area overhead than muxscan cell.
- Requires two or three high-speed scan clock signals based on the variation of LSSD used.
- Additional I/Os are required for the scan control signals.
- More suitable for latch-based designs.

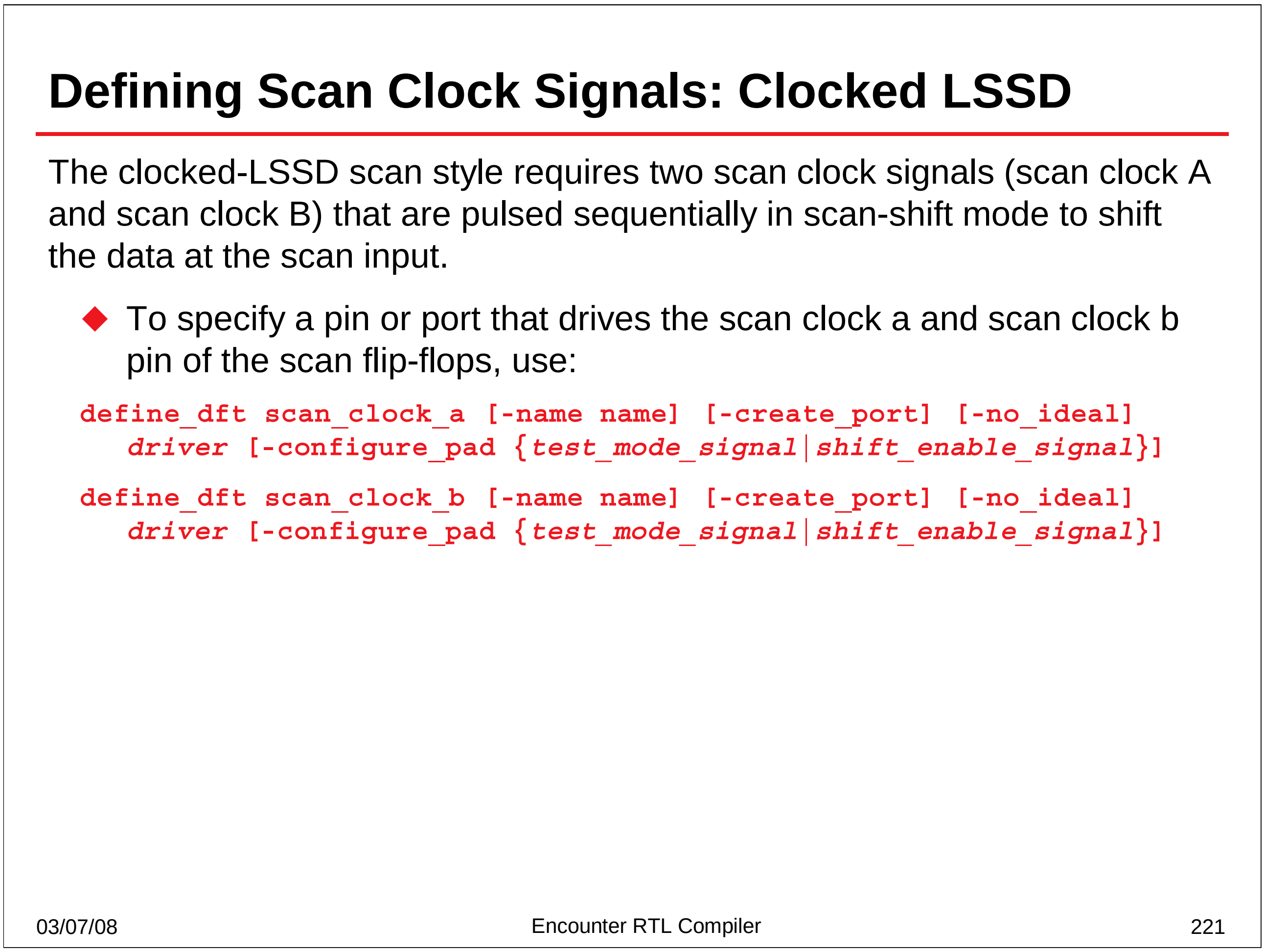


Defining Shift Enable Signals

Specify a pin or port that drives the *shift_enable* pin of the scan flip-flops. Use the *define_dft shift_enable* constraint:

```
define_dft shift_enable [-name test_signal] [-no_ideal]
   [-configure_pad] -active {low|high} [-create_port]
   port_name
```

- The muxed scan style uses shift-enable signals to switch scan flip-flops from system mode to scan-shift mode. Define these signals before running the DFT rule checker.
- You can define either one scan enable per scan chain, or one scan enable for all the scan chains.



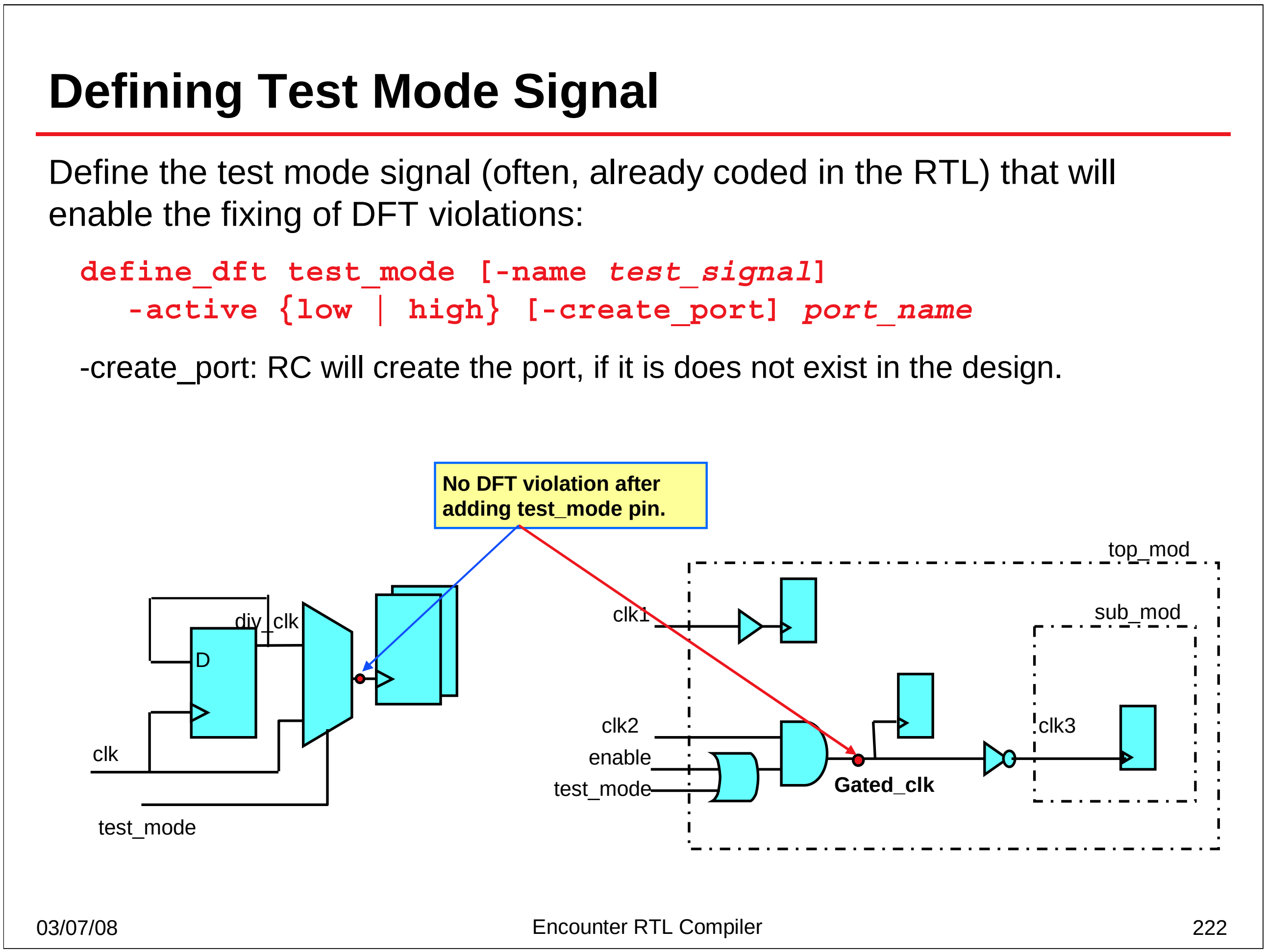
Defining Scan Clock Signals: Clocked LSSD

The clocked-LSSD scan style requires two scan clock signals (scan clock A and scan clock B) that are pulsed sequentially in scan-shift mode to shift the data at the scan input.

◆ To specify a pin or port that drives the scan clock a and scan clock b pin of the scan flip-flops, use:

```
define_dft scan_clock_a [-name name] [-create_port] [-no_ideal]
   driver [-configure_pad {test_mode_signal|shift_enable_signal}]

define_dft scan_clock_b [-name name] [-create_port] [-no_ideal]
   driver [-configure_pad {test_mode_signal|shift_enable_signal}]
```

When you define a scan clock signal as ideal, the DFT engine sets the *ideal_network* attribute to true on the pin or port for the scan clock signal. If you redefine the signal, this *ideal_network* attribute is **not** reset automatically.



Defining Test Mode Signal

Define the test mode signal (often, already coded in the RTL) that will enable the fixing of DFT violations:

```
define_dft test_mode [-name test_signal]
   -active {low | high} [-create_port] port_name
```

-create_port: RC will create the port, if it is does not exist in the design.

No DFT violation after adding test_mode pin.

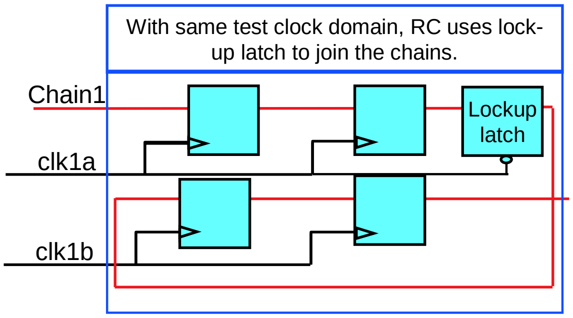
Use test-mode signals to bypass internally-generated clocks and enabled gated clocks and to inactivate asynchronous set or reset signals to pins of flip-flops during the scan-shift mode.

Internal Test Clock

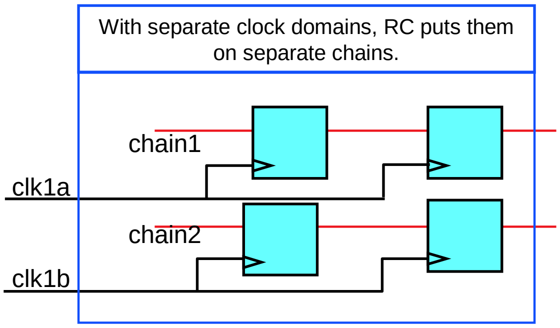
Define the internal clock branches as separate test clocks when it is an unbalanced clock-tree.

```
define_dft test_clock
   -name test_clock
   -domain test_clock_domain
   -period integer
   [-divide_period integer]
   [-rise integer]
   [-divide_rise integer]
   [-fall integer]
   [-divide_fall integer]
   [-controllable]
   { pin_name [pin_name] ...}
```

If you use the same test clock domain for both test clocks, you allow all flip-flops to be mixed in one scan chain. RC automatically adds a lock-up latch for all clocks in the same domain.



With same test clock domain, RC uses lock-up latch to join the chains.



With separate clock domains, RC puts them on separate chains.

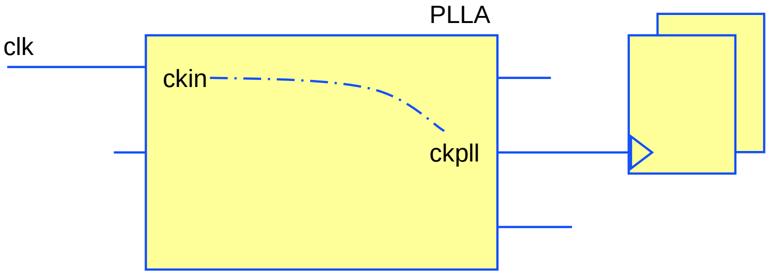
To automatically identify the output of all mapped combinational logic as internal test clocks in the corresponding root test-clock domain, use:

```
set_attr dft_identify_internal_test_clocks [true/false] /
```

Defining DFT Controllable Constraint

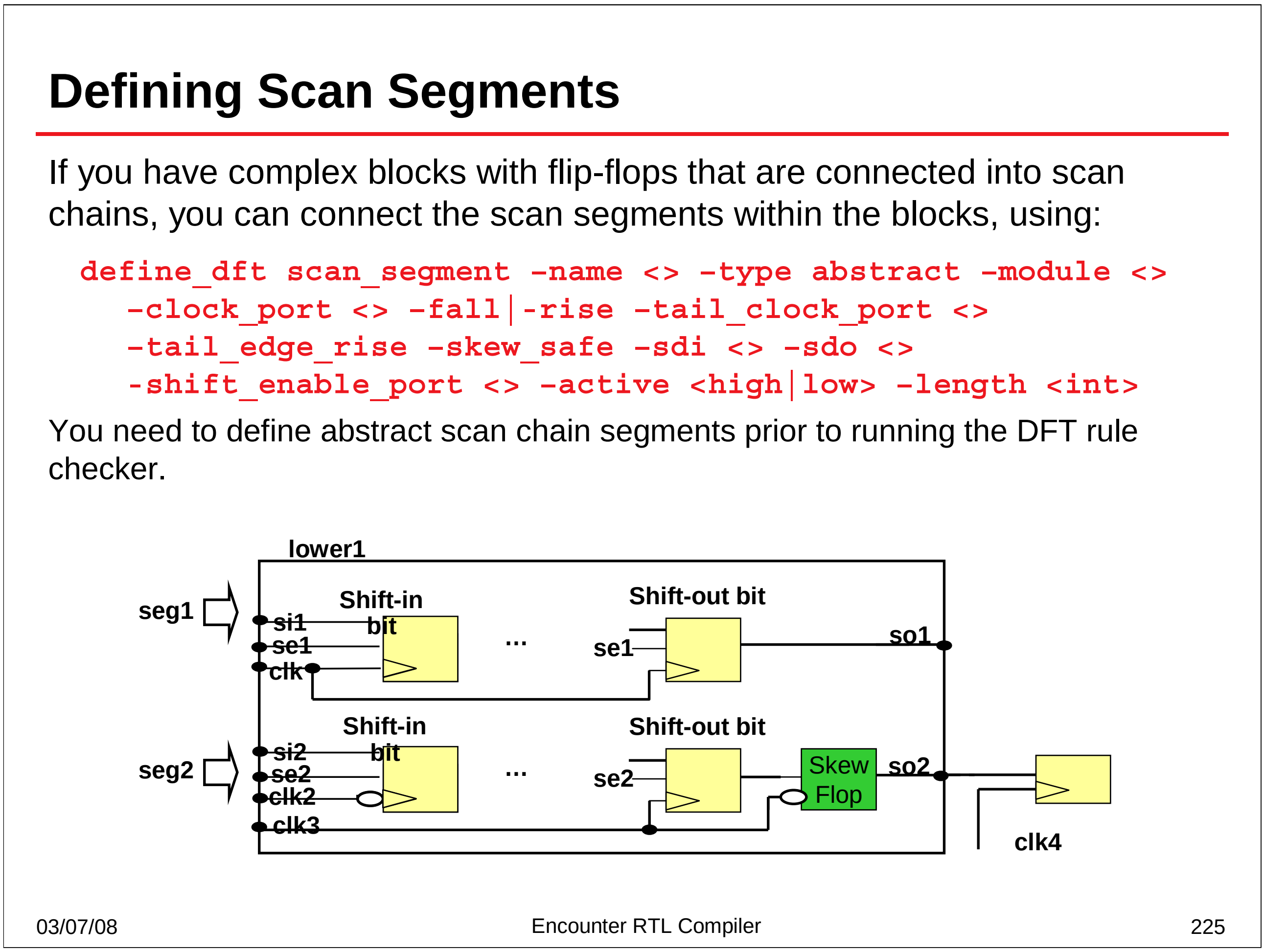
To enable propagation of the DFT rule checks across the specified pins of a black box module, such as a PLL, use this command:

```
set_attr dft_controllable "instance/fromPin
[non_inverting|inverting]" instance/toPin
```



Example

```
set_attr dft_controllable "PLLA/ckin inverting"
PLLA/clkpll
```



Defining Scan Segments

If you have complex blocks with flip-flops that are connected into scan chains, you can connect the scan segments within the blocks, using:

```
define_dft scan_segment -name <> -type abstract -module <>
   -clock_port <> -fall|-rise -tail_clock_port <>
   -tail_edge_rise -skew_safe -sdi <> -sdo <>
   -shift_enable_port <> -active <high|low> -length <int>
```

You need to define abstract scan chain segments prior to running the DFT rule checker.

03/07/08 Encounter RTL Compiler 225

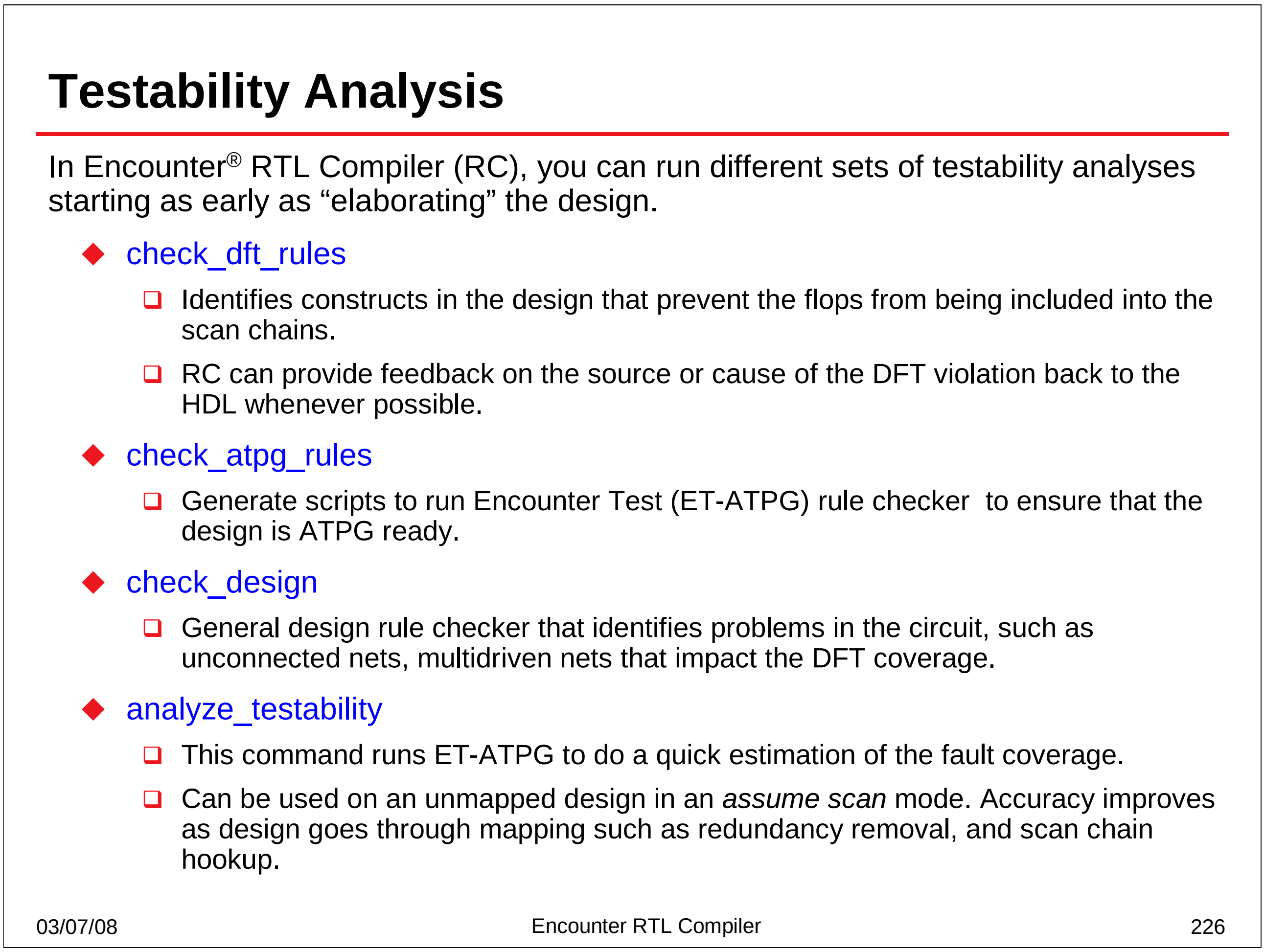
Example

```
define_dft scan_segment -type abstract -module core \
  -sdi subports_in/SI2 -sdo subports_out/SO2 \
  -clock_port subports_in/clk2 -rise \
  -tail_clock_port subports_in/clk3 -tail_edge_fall \
  -shift_enable_port subports_in/SE2 -active low \
  -length 3 -skew_safe -name abstract2
```

In this example, the scan chain has a sequential length of 3. The scan chain uses the rising edge of *clk2* as the shift-in scan clock, and the falling edge of *clk3* as the shift-out scan clock.

You can also define scan segments of type fixed (map, no reordering), floating (map and reorder), or preserve (no remap, or no reordering) and define the elements within them.

The -*skew_safe* option indicates whether the abstract segment has a data lockup element connected at the end of its scan chain. This option applies only if -*type* is set to *abstract*.



Testability Analysis

In Encounter® RTL Compiler (RC), you can run different sets of testability analyses starting as early as “elaborating” the design.

- check_dft_rules
  - ❑ Identifies constructs in the design that prevent the flops from being included into the scan chains.
  - ❑ RC can provide feedback on the source or cause of the DFT violation back to the HDL whenever possible.
- check_atpg_rules
  - ❑ Generate scripts to run Encounter Test (ET-ATPG) rule checker to ensure that the design is ATPG ready.
- check_design
  - ❑ General design rule checker that identifies problems in the circuit, such as unconnected nets, multidriven nets that impact the DFT coverage.
- analyze_testability
  - ❑ This command runs ET-ATPG to do a quick estimation of the fault coverage.
  - ❑ Can be used on an unmapped design in an *assume scan* mode. Accuracy improves as design goes through mapping such as redundancy removal, and scan chain hookup.

ATPG: Automatic Test Patten Generator

Checking DFT Rules

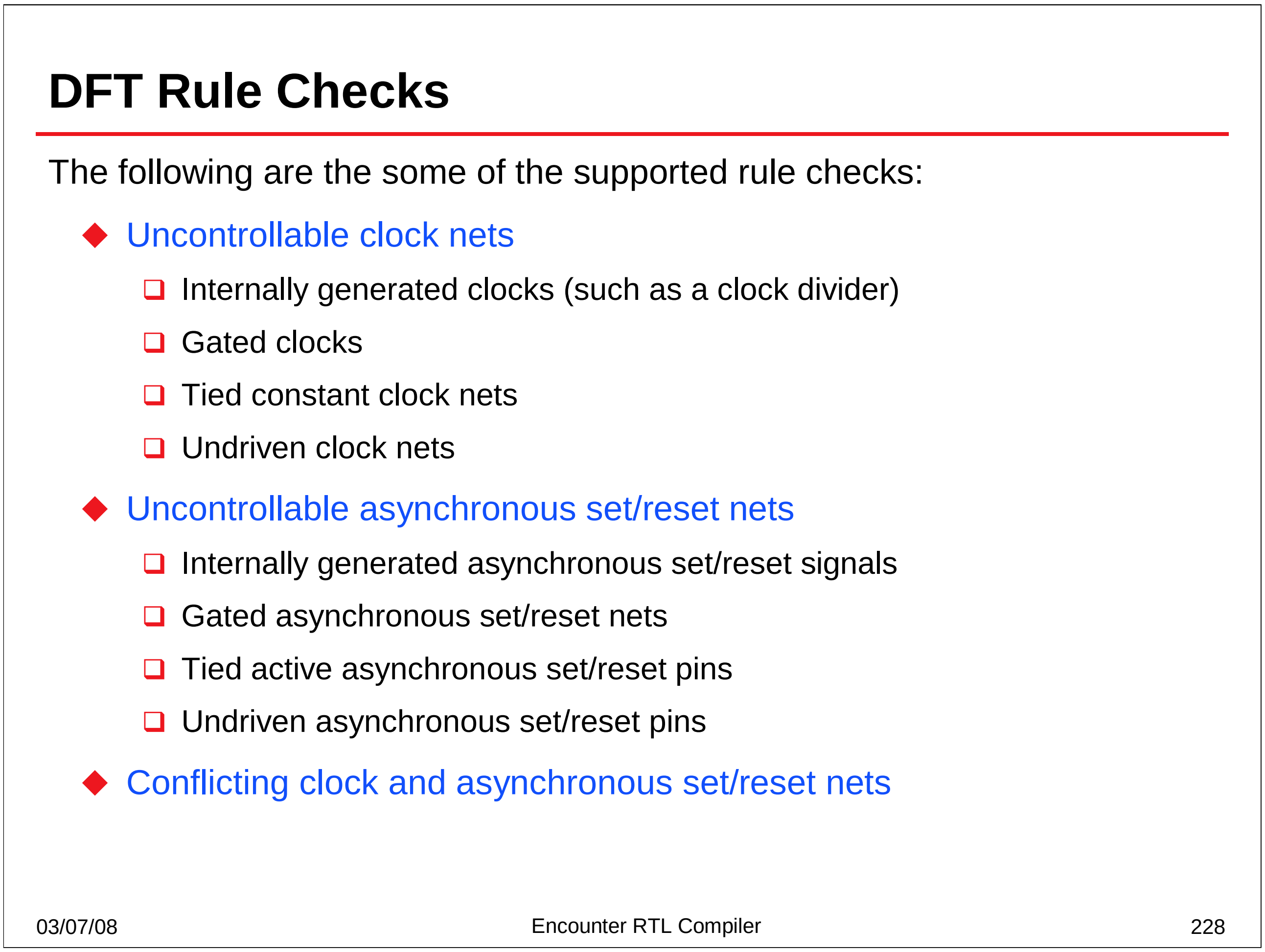
To check for any DFT violations, use this command:

```
check_dft_rules
```

Report

```
Warning : DFT Clock Rule Violation. [DFT-301]
: # 0: internal or gated clock signal in module ’top’, net:’ Iclk’, pin ’g1/z’
Effective fanin cone: clk, en
Warning : DFT Async Rule Violation. [DFT-302]
: # 1: async signal driven by a sequential element in module ’top’, net:’
Iset’, pin ’Iset_reg/q’
Effective fanin cone: Iset_reg/q
Violation # 0 affects 4 registers
Violation # 1 affects 4 registers
Note - a register may be violating multiple DFT rules
Total number of Test Clock Domains: 1
DFT Test Clock Domain: clk
Test Clock ’clk’ (Positive edge) has 1 registers
Test Clock ’clk’ (Negative edge) has 0 registers
Number of user specified non-Scan registers: 0
Number of registers that fail DFT rules: 4
Number of registers that pass DFT rules: 1
Percentage of total registers that are scanable: 20%
```

If a flip-flop passes the DFT rule checks, automatic test pattern generated (ATPG) data can safely be shifted in and out of the scan flip-flop during the scan shift cycle of test mode. If a flip-flop violates the DFT rule checks, it is marked for exclusion from the scan chain. The fewer the violating flops, the higher the fault coverage.



DFT Rule Checks

The following are the some of the supported rule checks:

- Uncontrollable clock nets
  - ❑ Internally generated clocks (such as a clock divider)
  - ❑ Gated clocks
  - ❑ Tied constant clock nets
  - ❑ Undriven clock nets
- Uncontrollable asynchronous set/reset nets
  - ❑ Internally generated asynchronous set/reset signals
  - ❑ Gated asynchronous set/reset nets
  - ❑ Tied active asynchronous set/reset pins
  - ❑ Undriven asynchronous set/reset pins
- Conflicting clock and asynchronous set/reset nets

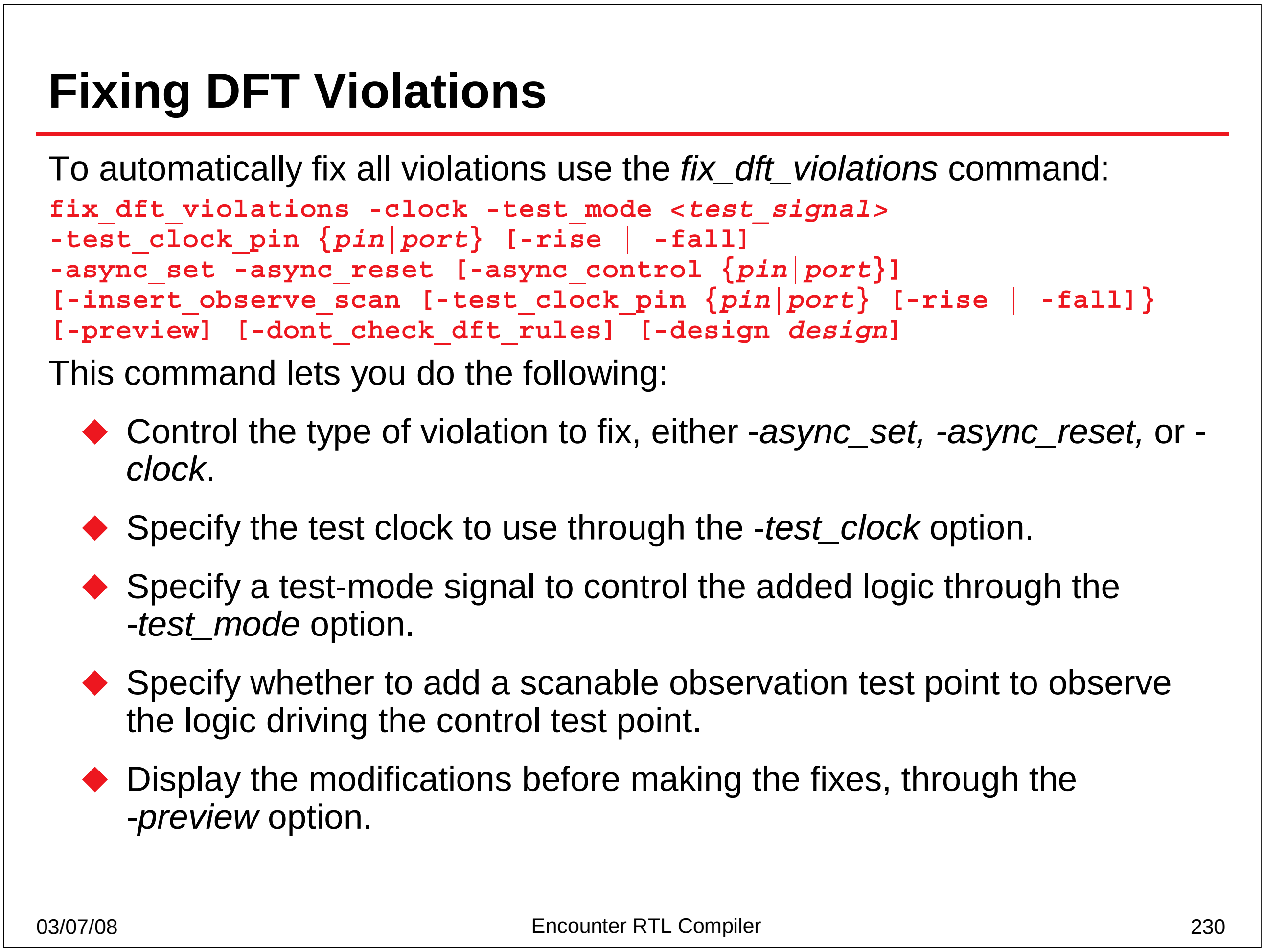
Reporting Scan Flops

To report the scanable status of the flip-flops after running the DFT rule checker, use:

```
report dft_registers
```

This example shows the report for a design with an internally driven clock signal and an asynchronous set signal.

```
Reporting registers that pass DFT rules
Iset_reg PASS; Test clock: clk/rise
Reporting registers that fail DFT rules
out_reg_0 FAIL; violations: clock #(0 ) async set #(1 )
out_reg_1 FAIL; violations: clock #(0 ) async set #(1 )
out_reg_2 FAIL; violations: clock #(0 ) async set #(1 )
out_reg_3 FAIL; violations: clock #(0 ) async set #(1 )
Total registers that pass DFT rules: 1
Total registers that fail DFT rules: 4
```



Fixing DFT Violations

To automatically fix all violations use the *fix_dft_violations* command:

```
fix_dft_violations -clock -test_mode <test_signal>
-test_clock_pin {pin|port} [-rise | -fall]
-async_set -async_reset [-async_control {pin|port}]
[-insert_observe_scan [-test_clock_pin {pin|port} [-rise | -fall]}
[-preview] [-dont_check_dft_rules] [-design design]
```

This command lets you do the following:

- Control the type of violation to fix, either -*async_set, -async_reset,* or -*clock*.
- Specify the test clock to use through the -*test_clock* option.
- Specify a test-mode signal to control the added logic through the -*test_mode* option.
- Specify whether to add a scanable observation test point to observe the logic driving the control test point.
- Display the modifications before making the fixes, through the -*preview* option.

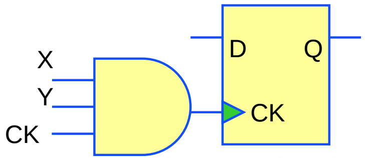
You can selectively fix any DFT violation by specifying its violation identifier (VID). Use:

```
fix_dft_violations -violations <VID>
```

The violation identifier is printed in the log file during *check_dft_rules*. It has the form *vid_<n>_[async* | *clock* | *abs]*, where *n* is a positive integer.

Example: Fixing Clock Violations in a Circuit

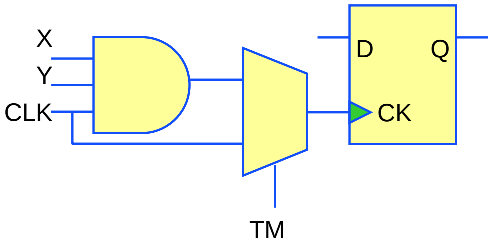
The example shows the auto-fixing of DFT clock violations using the *fix_dft_violations* command.



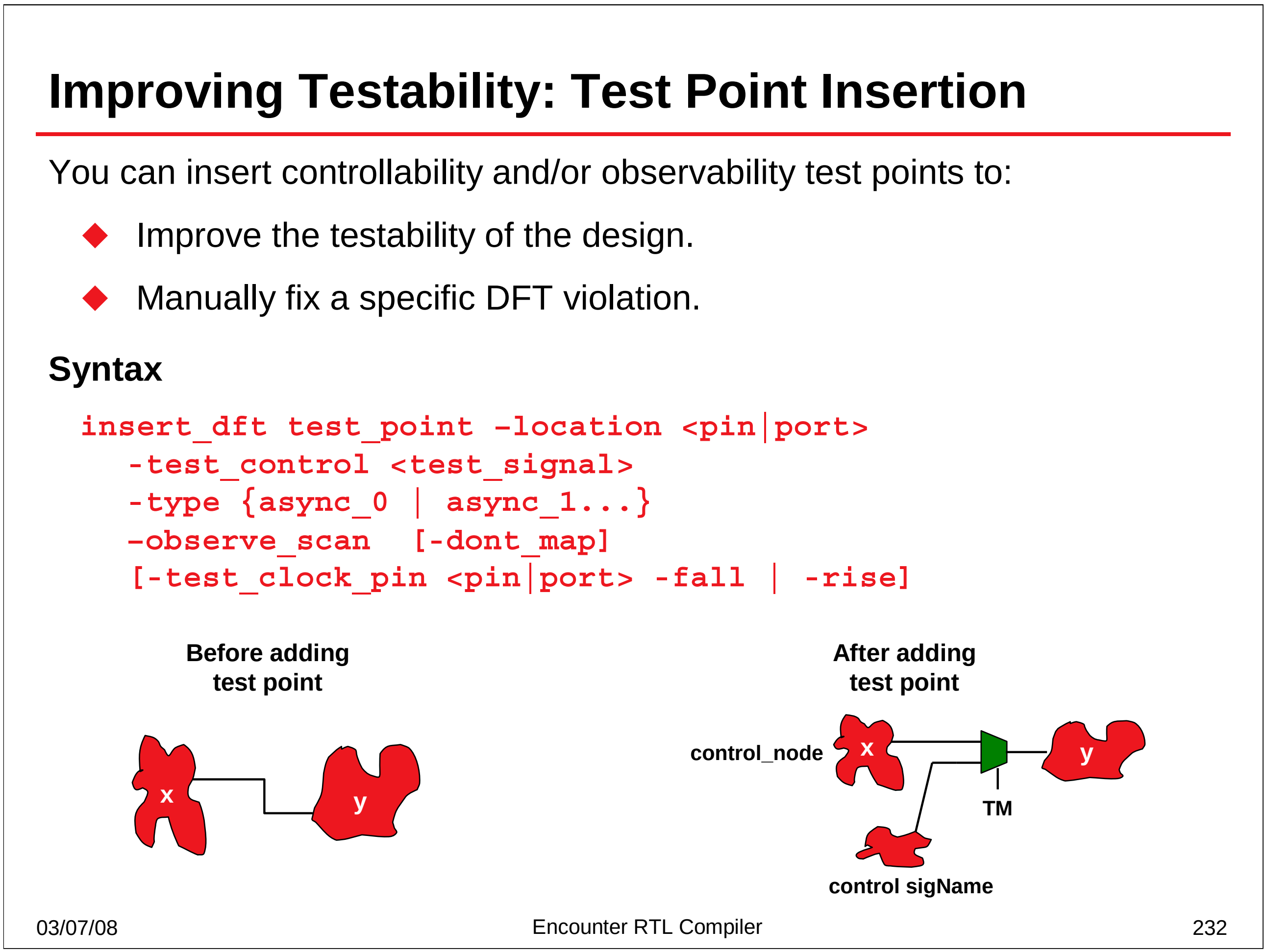
Before: Clock is uncontrollable

The following commands specify the violation to fix and the appropriate test point insertion.

```
define_dft test_mode -active high TM
fix_dft_violations -clock -test_mode TM -test_clock_pin CLK
```



After: Gated Clock is controllable



Improving Testability: Test Point Insertion

You can insert controllability and/or observability test points to:

- Improve the testability of the design.
- Manually fix a specific DFT violation.

Syntax

```
insert_dft test_point -location <pin|port>
   -test_control <test_signal>
   -type {async_0 | async_1...}
   -observe_scan  [-dont_map]
   [-test_clock_pin <pin|port> -fall | -rise]
```

Before adding
test point
x
y
After adding
test point
control_node
x
y
TM
control sigName

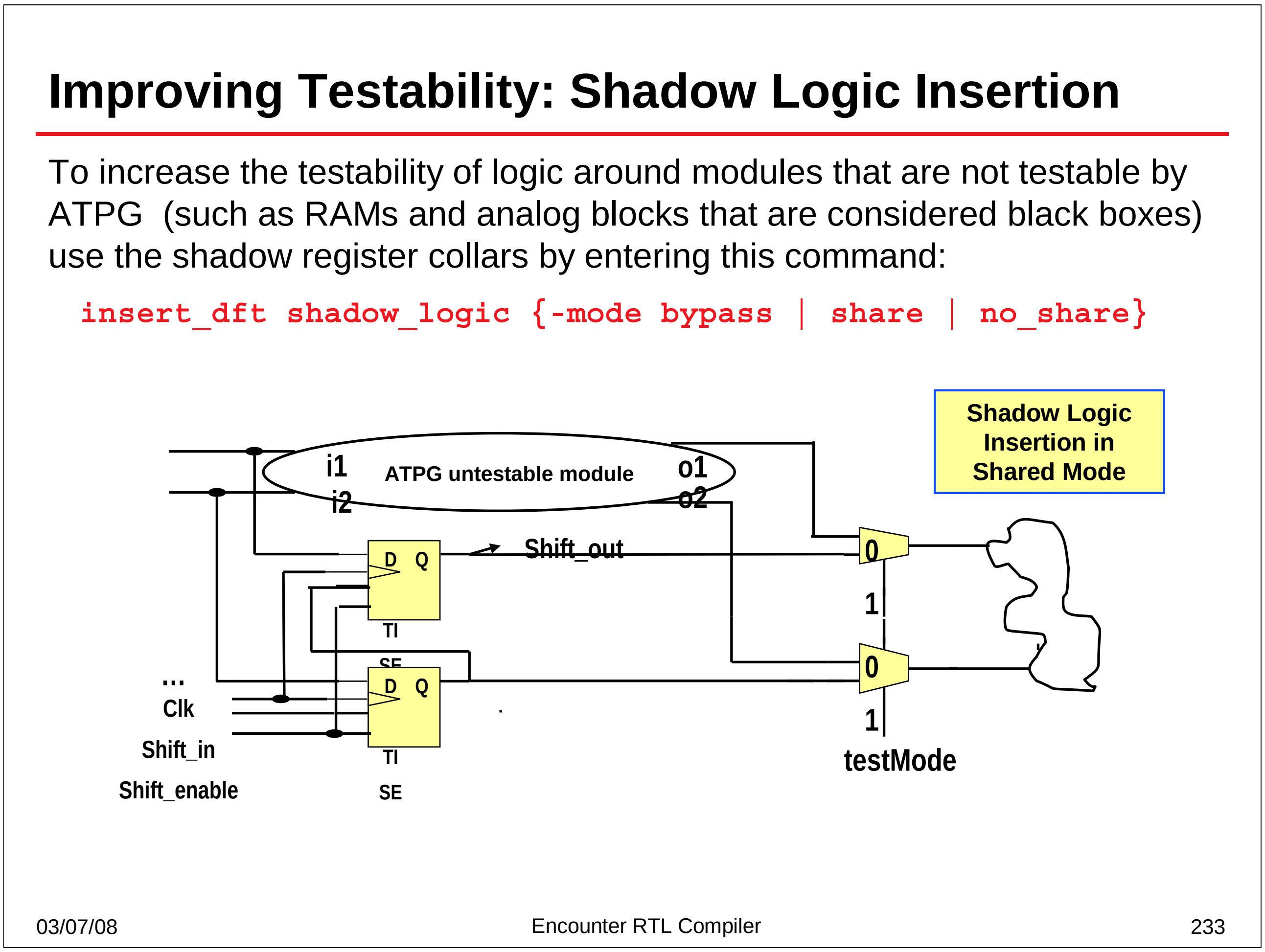
03/07/08 Encounter RTL Compiler 232

Control test points always require the specification of a test-mode signal. Test points that use scannable flip-flops to observe or control a node always require a test-clock signal.

For all of the scannable test points, you need to run *check_dft_rules* after the test point is inserted. You also need to run incremental synthesis to remap the control or observation flip-flop to a scan flop prior to configuring the scan chains in the design.

To insert a user-defined test point at the specified location, and hook it up to the specified pins, use the following command:

```
insert_dft user_test_point -location {pin|port|subport}
  -cell {design|subdesign|libcell}
  -cfi {pin|port} -cfo {pin|port}
  -connect string [-connect string]...
  -name name
```



Improving Testability: Shadow Logic Insertion

To increase the testability of logic around modules that are not testable by ATPG (such as RAMs and analog blocks that are considered black boxes) use the shadow register collars by entering this command:

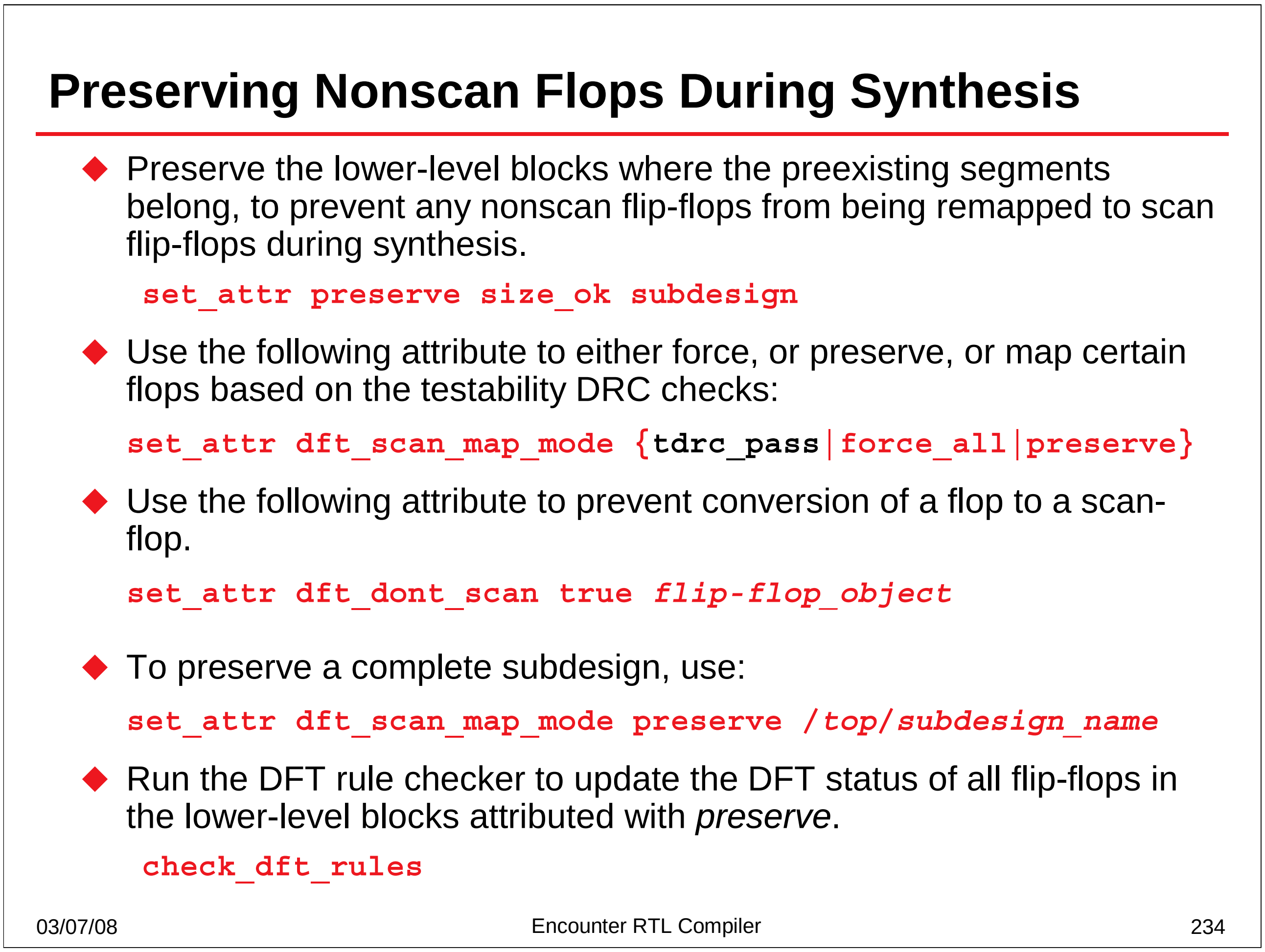
```
insert_dft shadow_logic {-mode bypass | share | no_share}
```

Shadow Logic Insertion in Shared Mode

The command can insert two basic types of DFT shadow logic around a particular instance: bypass and scannable logic. Each shadow logic flip-flop can implement one control point and one observation point at the same time.

The command syntax is as follows:

```
insert_dft shadow_logic -around instance
  [-test_control test_signal] {-mode bypass |
  {no_share -test_clock_pin {port|pin} [-fall|-rise]}
  {share -test_clock_pin {port|pin}[-fall|-rise] }}
  [-exclude pins | -only pins ] [-group pins]...
  [-balance] [-dont_map] [-preview]
```



Preserving Nonscan Flops During Synthesis

- Preserve the lower-level blocks where the preexisting segments belong, to prevent any nonscan flip-flops from being remapped to scan flip-flops during synthesis.

```
set_attr preserve size_ok subdesign
```

- Use the following attribute to either force, or preserve, or map certain flops based on the testability DRC checks:

```
set_attr dft_scan_map_mode {tdrc_pass|force_all|preserve}
```

- Use the following attribute to prevent conversion of a flop to a scan-flop.

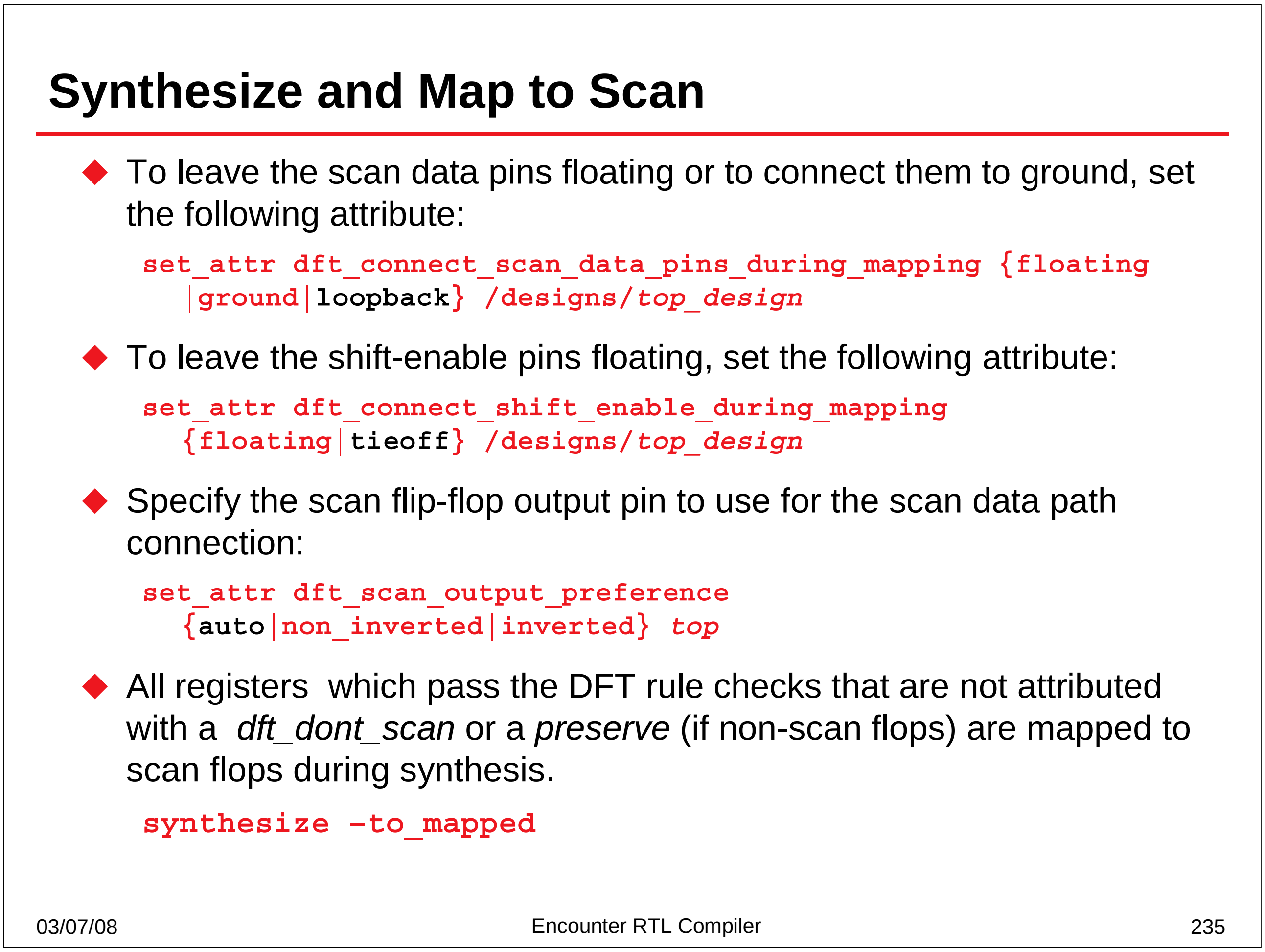
```
set_attr dft_dont_scan true flip-flop_object
```

- To preserve a complete subdesign, use:

```
set_attr dft_scan_map_mode preserve /top/subdesign_name
```

- Run the DFT rule checker to update the DFT status of all flip-flops in the lower-level blocks attributed with *preserve*.

```
check_dft_rules
```



Synthesize and Map to Scan

- To leave the scan data pins floating or to connect them to ground, set the following attribute:

```
set_attr dft_connect_scan_data_pins_during_mapping {floating
   |ground|loopback} /designs/top_design
```

- To leave the shift-enable pins floating, set the following attribute:

```
set_attr dft_connect_shift_enable_during_mapping
   {floating|tieoff} /designs/top_design
```

- Specify the scan flip-flop output pin to use for the scan data path connection:

```
set_attr dft_scan_output_preference
   {auto|non_inverted|inverted} top
```

- All registers which pass the DFT rule checks that are not attributed with a *dft_dont_scan* or a *preserve* (if non-scan flops) are mapped to scan flops during synthesis.

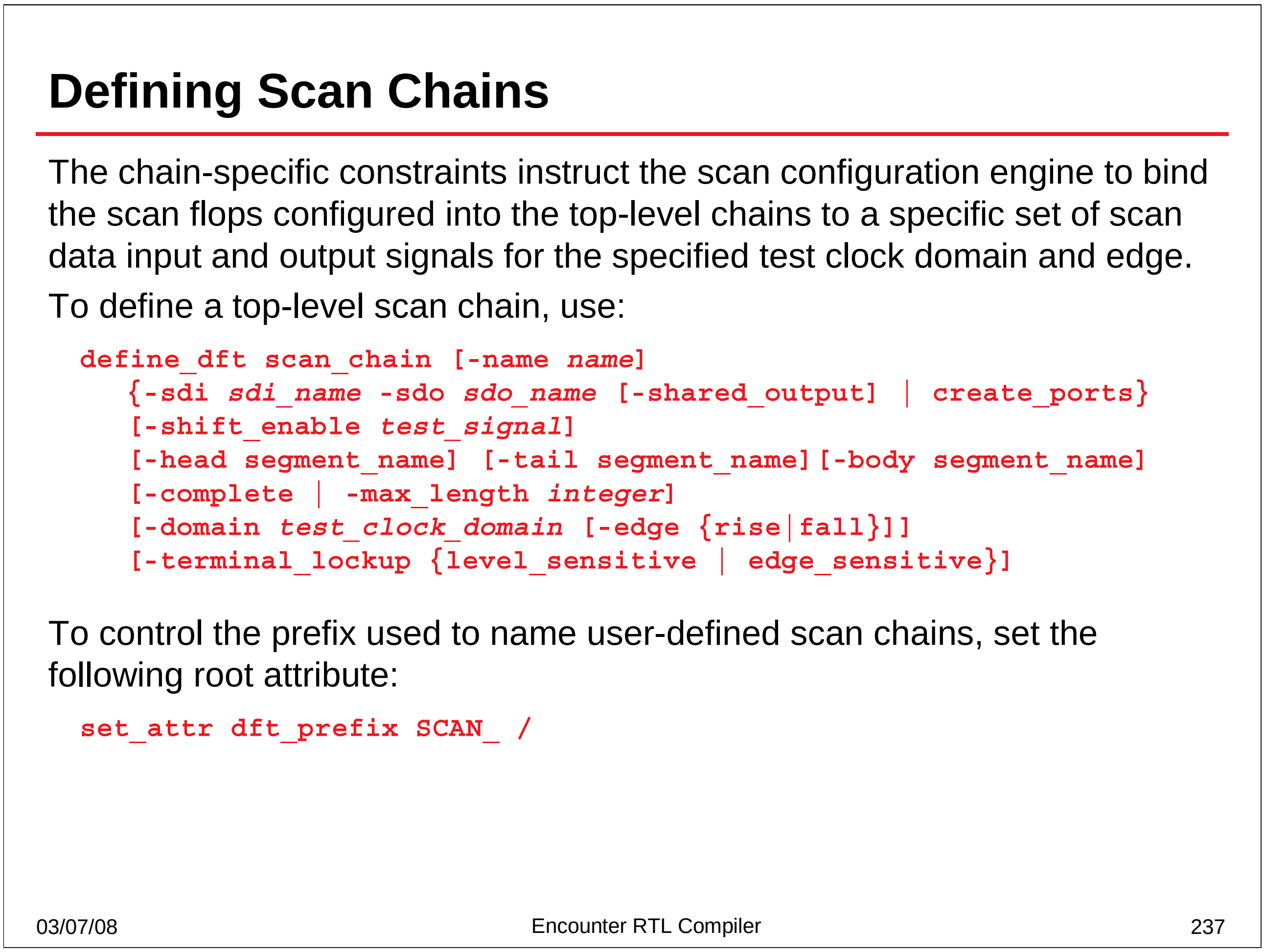
```
synthesize -to_mapped
```

Fixing DFT Violations on a Scan-Mapped Design

- Following initial synthesis, you can still fix the DFT rule violations in the mapped netlist by running the *fix_dft_violations* command. To convert the remaining flip-flops that now pass the DFT rule checks to scan flip-flops, run this command:

```
replace_scan
```

- All nonscan registers that pass the DFT rule checks and are not attributed with a *dft_dont_scan* or a preserve (if nonscan flops) will be remapped to scan flops.



Defining Scan Chains

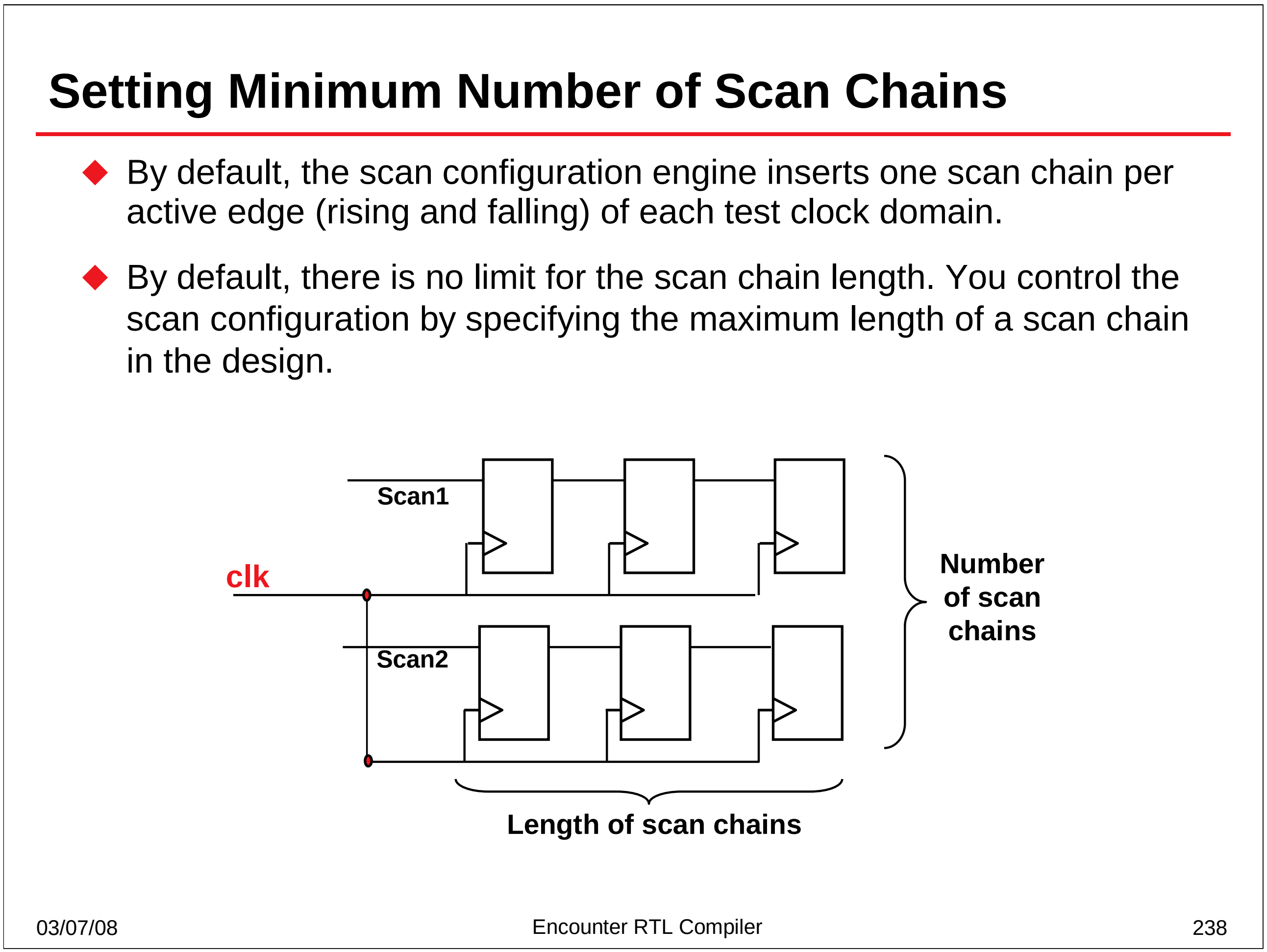
The chain-specific constraints instruct the scan configuration engine to bind the scan flops configured into the top-level chains to a specific set of scan data input and output signals for the specified test clock domain and edge.

To define a top-level scan chain, use:

```
define_dft scan_chain [-name name]
   {-sdi sdi_name -sdo sdo_name [-shared_output] | create_ports}
   [-shift_enable test_signal]
   [-head segment_name] [-tail segment_name][-body segment_name]
   [-complete | -max_length integer]
   [-domain test_clock_domain [-edge {rise|fall}]]
   [-terminal_lockup {level_sensitive | edge_sensitive}]
```

To control the prefix used to name user-defined scan chains, set the following root attribute:

```
set_attr dft_prefix SCAN_ /
```



Setting Minimum Number of Scan Chains

- By default, the scan configuration engine inserts one scan chain per active edge (rising and falling) of each test clock domain.
- By default, there is no limit for the scan chain length. You control the scan configuration by specifying the maximum length of a scan chain in the design.

- To specify the minimum number of scan chains to be created, set the following design attribute:

```
set_attribute dft_min_number_of_scan_chains integer /designs/top
```

- To specify the maximum length of any scan chain, set the following design attribute:

```
set_attribute dft_max_length_of_scan_chains integer /designs/top
```

Example of a Minimum Number of Scan Chains

RTL

- 300 rising-edge scan flip-flops in test clock domain: clk1
- 100 falling-edge scan flip-flops in test clock domain: clk2

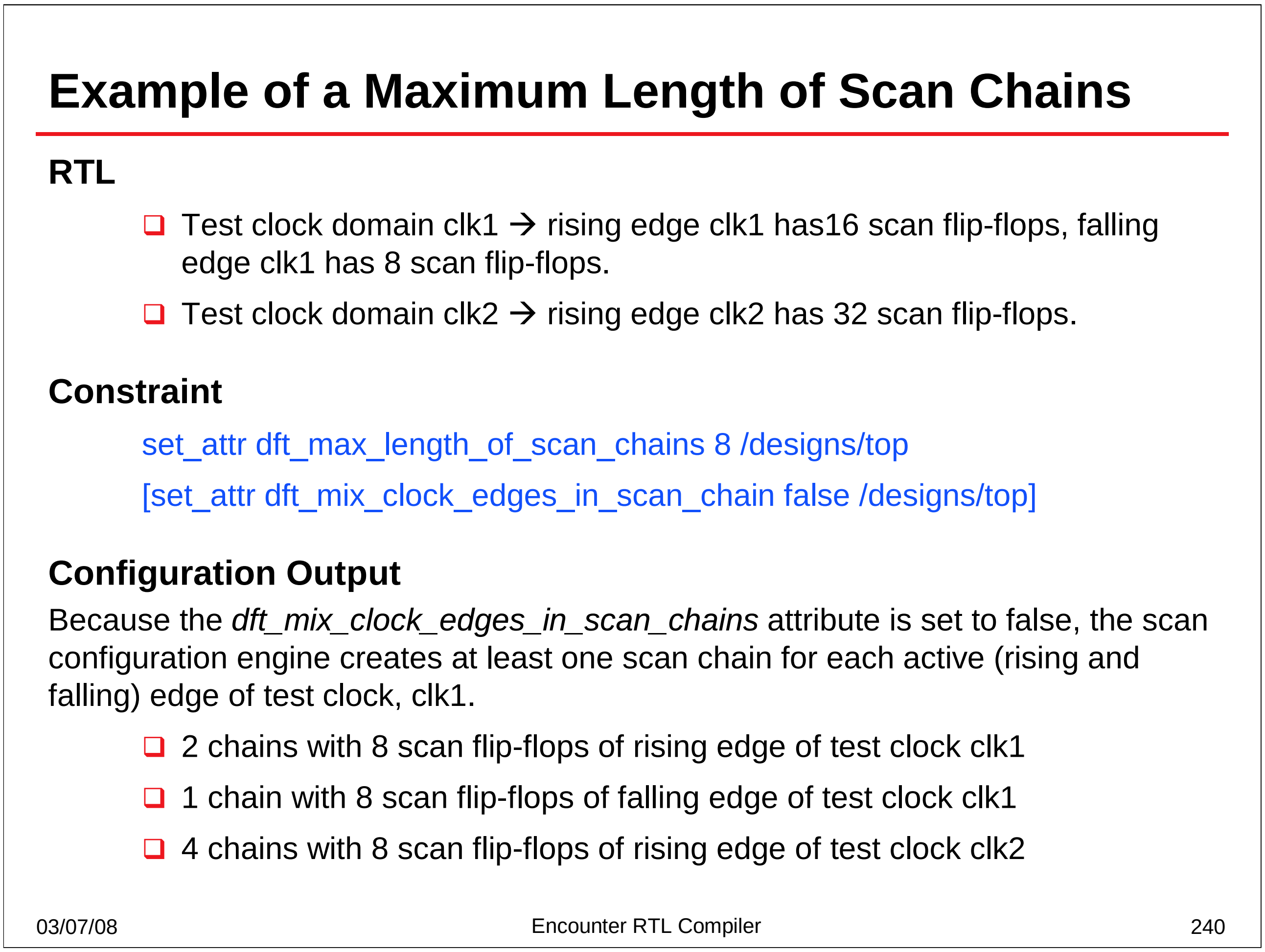
Constraint

```
set_attr dft_min_number_of_scan_chains 5 /designs/top
```

What Is the Configuration Output?

The scan configuration engine creates 5 scan chains whose lengths are balanced across the test clock domains:

- 4 scan chains of 75 scan flops each in test clock domain: clk1
- 1 scan chains of 100 scan flops each in test clock domain: clk2



Example of a Maximum Length of Scan Chains

RTL

- Test clock domain clk1 → rising edge clk1 has16 scan flip-flops, falling edge clk1 has 8 scan flip-flops.
- Test clock domain clk2 → rising edge clk2 has 32 scan flip-flops.

Constraint

```
set_attr dft_max_length_of_scan_chains 8 /designs/top
[set_attr dft_mix_clock_edges_in_scan_chain false /designs/top]
```

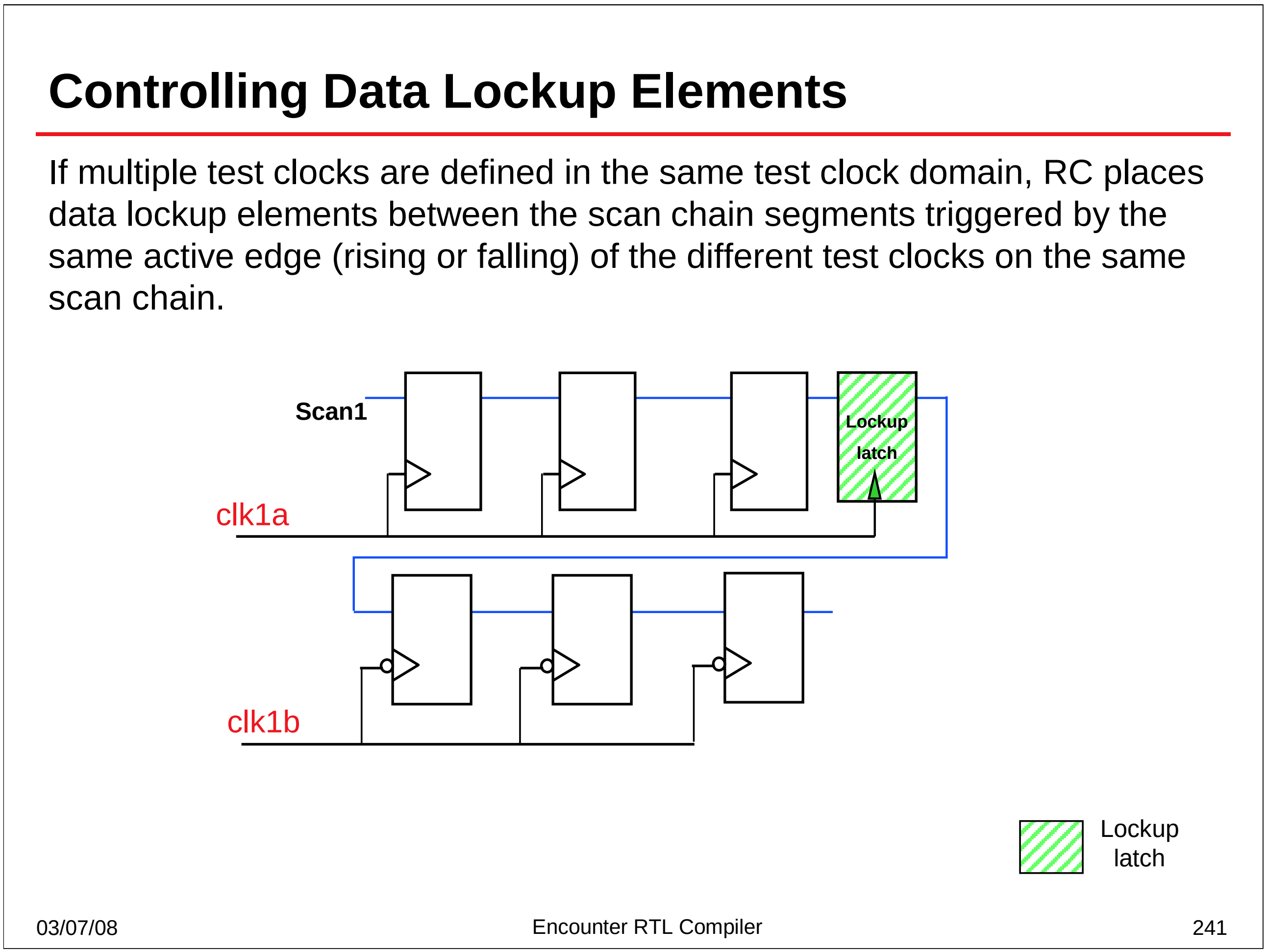
Configuration Output

Because the *dft_mix_clock_edges_in_scan_chains* attribute is set to false, the scan configuration engine creates at least one scan chain for each active (rising and falling) edge of test clock, clk1.

- 2 chains with 8 scan flip-flops of rising edge of test clock clk1
- 1 chain with 8 scan flip-flops of falling edge of test clock clk1
- 4 chains with 8 scan flip-flops of rising edge of test clock clk2

03/07/08 Encounter RTL Compiler 240

If the *dft_mix_clock_edges_in_scan_chains* attribute is set to *true*, then the configuration output is 3 scan chains of 8 flops with test clock *clk1* and 4 with 8 scan flops of test clock *clk2*, and the edges do not matter.



Controlling Data Lockup Elements

If multiple test clocks are defined in the same test clock domain, RC places data lockup elements between the scan chain segments triggered by the same active edge (rising or falling) of the different test clocks on the same scan chain.

▨ Lockup latch

03/07/08 Encounter RTL Compiler 241

- To allow the mixing of rising and falling edge-triggered scan flip-flops from the same test clock domain along the same scan chain, use this command:

  ```
  set_attribute dft_mix_clock_edges_in_scan_chain
  {true | false} top_design
  ```

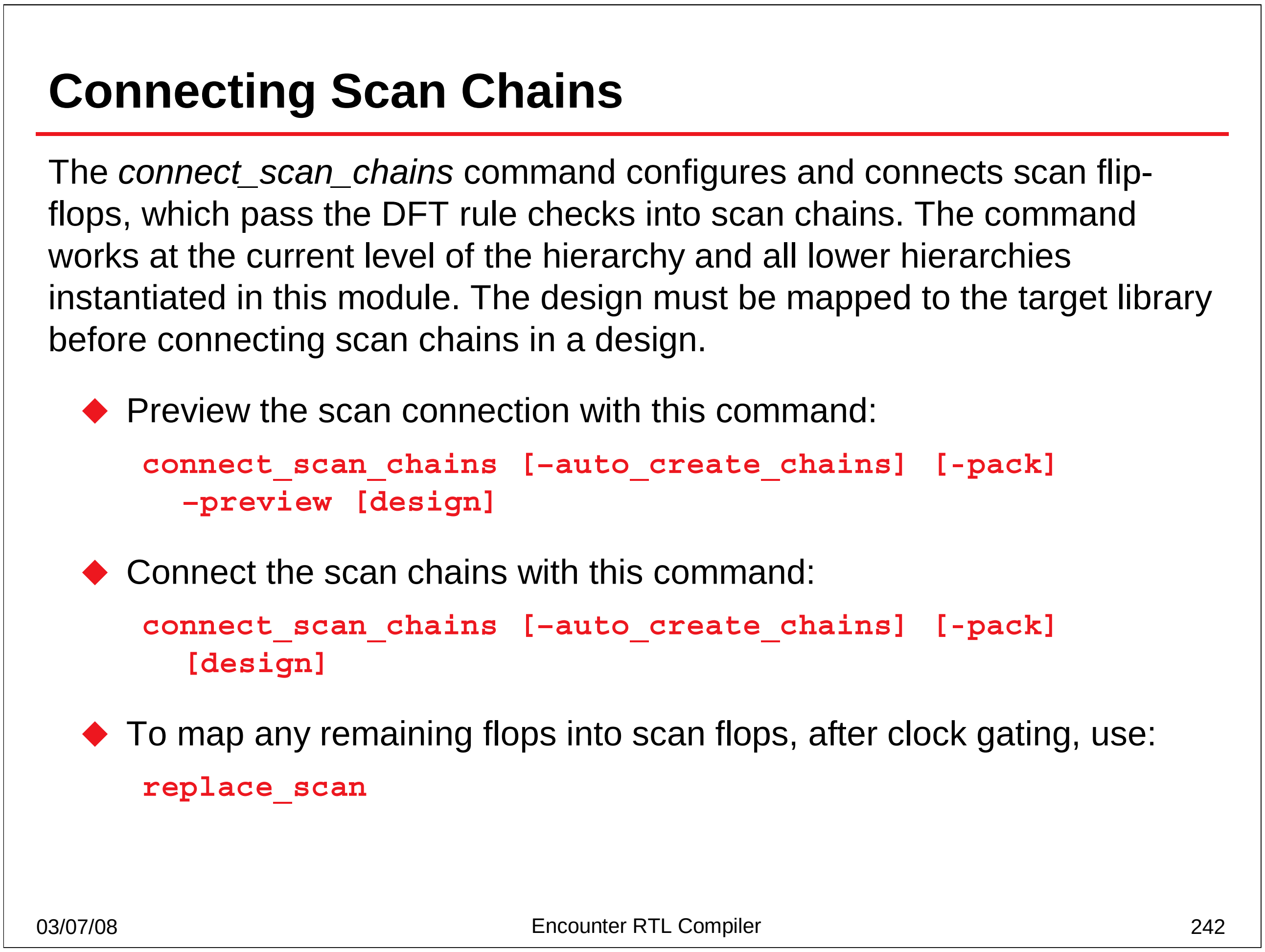
- To specify compatible clocks, whose related scan flip-flops can be merged into a single scan chain with lockup elements, use this command:

  ```
  set_compatible_test_clocks {-all | list_of_test_clocks}
  [-design design]
  ```

- To specify the default type of lockup element to include in all scan chains in the design, set the following design attribute:

  ```
  set_attr dft_lockup_element_type
  level_sensitive|edge_sensitive top_design
  ```

  The default value is *level_sensitive*.



Connecting Scan Chains

The *connect_scan_chains* command configures and connects scan flip-flops, which pass the DFT rule checks into scan chains. The command works at the current level of the hierarchy and all lower hierarchies instantiated in this module. The design must be mapped to the target library before connecting scan chains in a design.

- Preview the scan connection with this command:

```
connect_scan_chains [-auto_create_chains] [-pack]
  -preview [design]
```

- Connect the scan chains with this command:

```
connect_scan_chains [-auto_create_chains] [-pack]
  [design]
```

- To map any remaining flops into scan flops, after clock gating, use:

```
replace_scan
```

03/07/08 Encounter RTL Compiler 242

The *-pack* option packs the registers to meet the configuration for maximum length of scan chain, instead of the configuration for minimum number of scan chains.

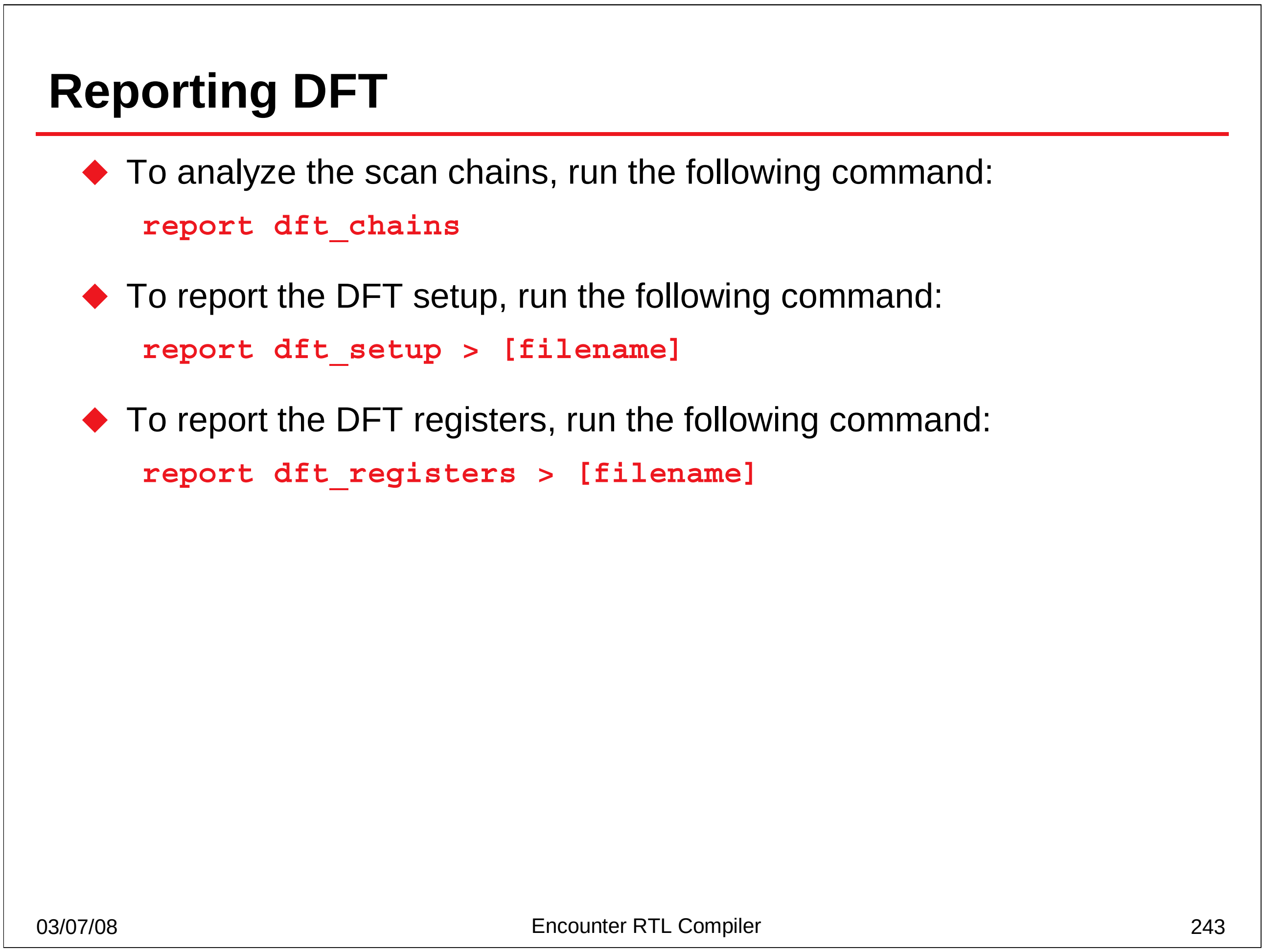
- The former approach serves better during bottom-up methodology to create proper scan segments.
- The latter approach better serves top-down methodology, or when you have precompiled blocks and you are configuring top-level scan chains.

Run Incremental Optimization

Connecting the scans can impact the timing. To fix any timing issues, run incremental optimization by entering this command:

```
synthesize -incremental
```

The *connect_scan_chains* command has a *-incremental* option.



Reporting DFT

- To analyze the scan chains, run the following command:

```
report dft_chains
```

- To report the DFT setup, run the following command:

```
report dft_setup > [filename]
```

- To report the DFT registers, run the following command:

```
report dft_registers > [filename]
```

Generating Output Scan Files

- To create a scanDEF interface file for scan chain reordering in the SoC Encounter™ system, use:

```
write_scandef > top_scan.def
```

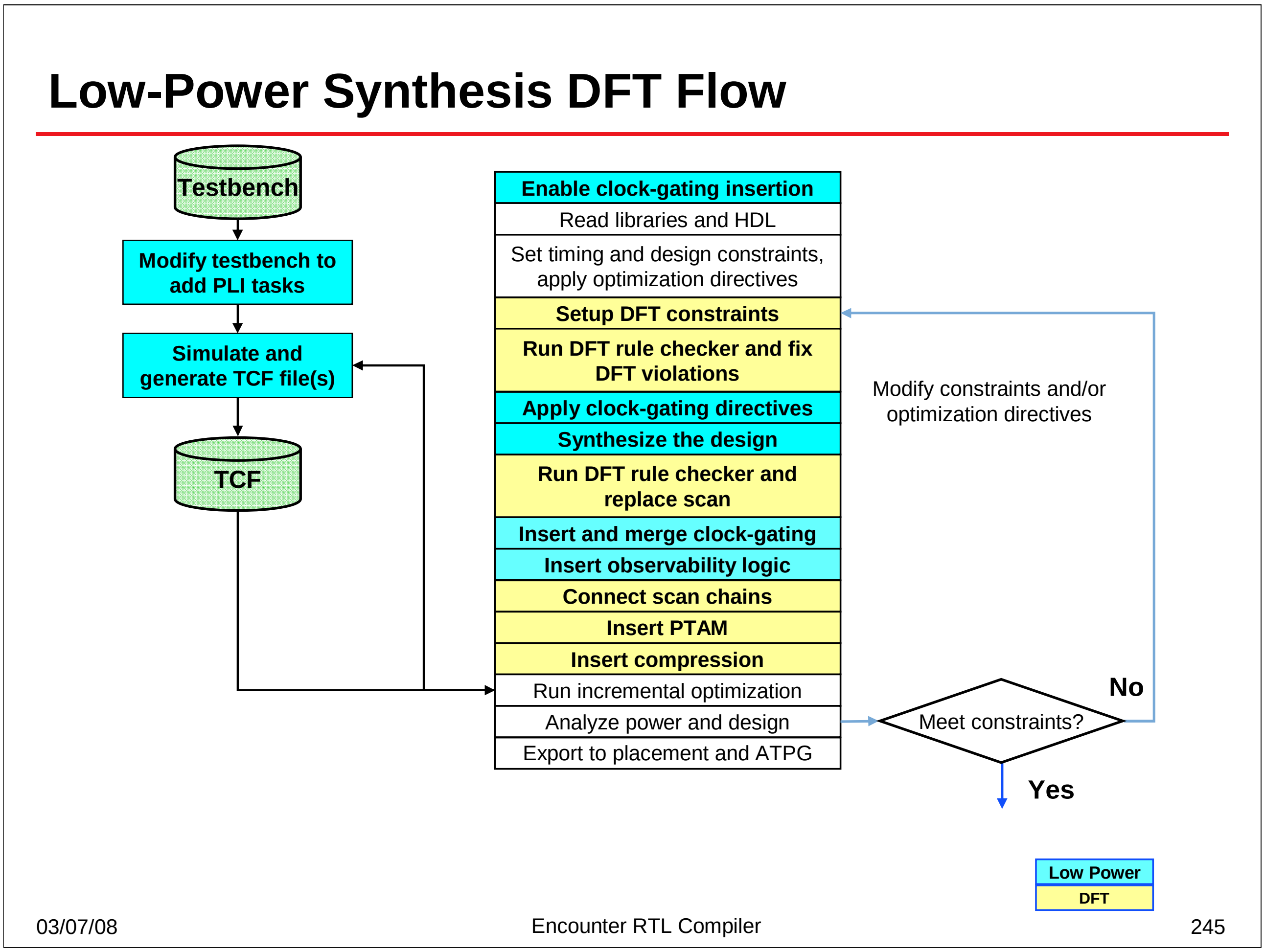
- To create an ATPG interface file, use:

```
write_atpg [-cadence | -stil | -mentor] > top_atpg.m
```

- To use your design as a subblock in another design, generate a scan chain abstraction model of your design by entering the following command:

```
write_dft_abstract_model > DFTModelName
```

  The abstraction models will be used by scan configuration when creating the top-level chains without requiring the netlist views of the lower level blocks.



Low-Power Synthesis DFT Flow

Testbench

Modify testbench to add PLI tasks

Simulate and generate TCF file(s)

TCF

Enable clock-gating insertion

Read libraries and HDL

Set timing and design constraints, apply optimization directives

Setup DFT constraints

Run DFT rule checker and fix DFT violations

Apply clock-gating directives

Synthesize the design

Run DFT rule checker and replace scan

Insert and merge clock-gating

Insert observability logic

Connect scan chains

Insert PTAM

Insert compression

Run incremental optimization

Analyze power and design

Export to placement and ATPG

Meet constraints?

No

Yes

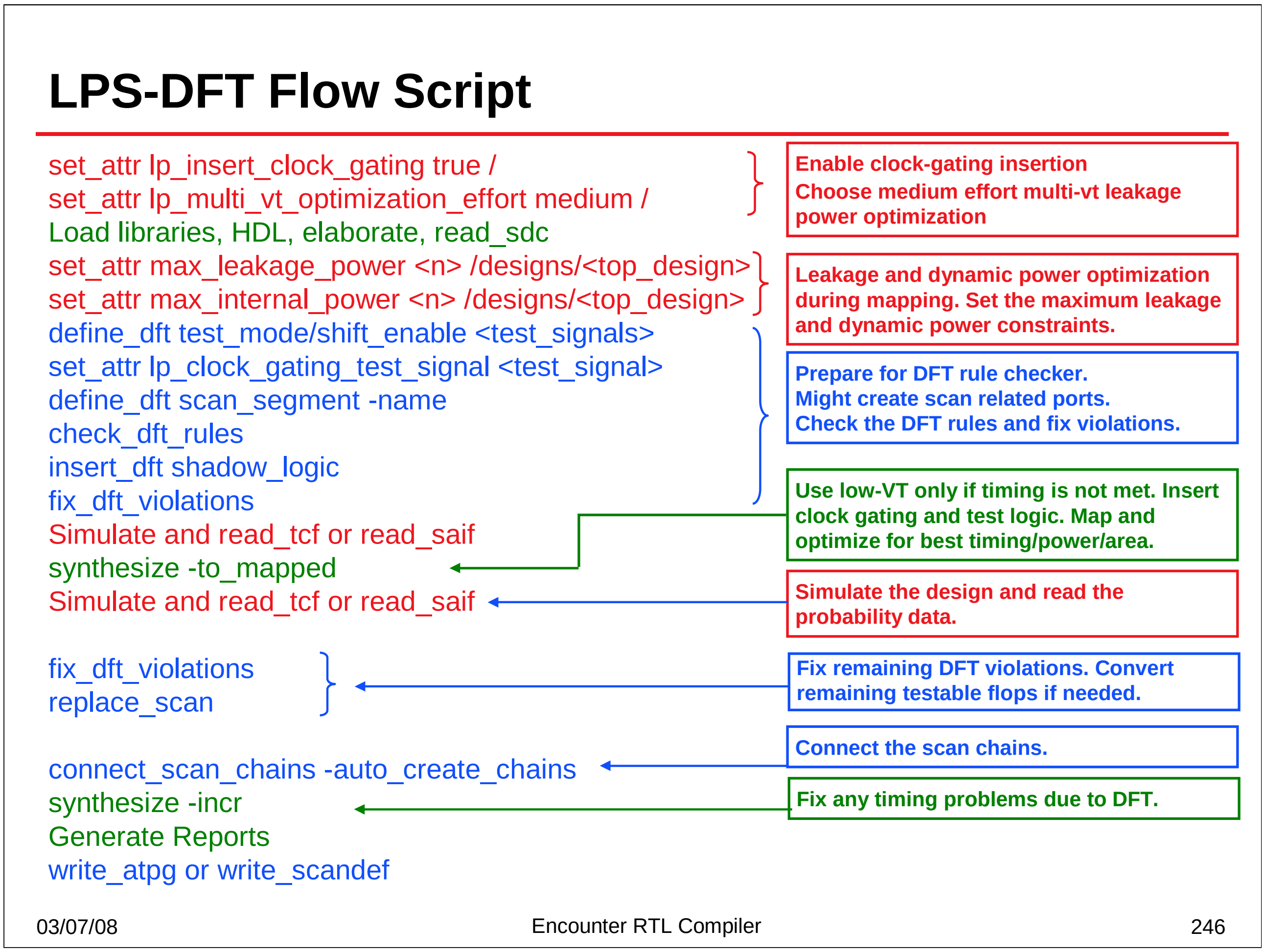
Modify constraints and/or optimization directives

Low Power
DFT

03/07/08 Encounter RTL Compiler 245

PLI: Programming Language Interface

TCF: Toggle Count Format



LPS-DFT Flow Script

```
set_attr lp_insert_clock_gating true /
set_attr lp_multi_vt_optimization_effort medium /
Load libraries, HDL, elaborate, read_sdc
set_attr max_leakage_power <n> /designs/<top_design>
set_attr max_internal_power <n> /designs/<top_design>
define_dft test_mode/shift_enable <test_signals>
set_attr lp_clock_gating_test_signal <test_signal>
define_dft scan_segment -name
check_dft_rules
insert_dft shadow_logic
fix_dft_violations
Simulate and read_tcf or read_saif
synthesize -to_mapped
Simulate and read_tcf or read_saif

fix_dft_violations
replace_scan

connect_scan_chains -auto_create_chains
synthesize -incr
Generate Reports
write_atpg or write_scandef
```

Enable clock-gating insertion
Choose medium effort multi-vt leakage power optimization

Leakage and dynamic power optimization during mapping. Set the maximum leakage and dynamic power constraints.

Prepare for DFT rule checker.
Might create scan related ports.
Check the DFT rules and fix violations.

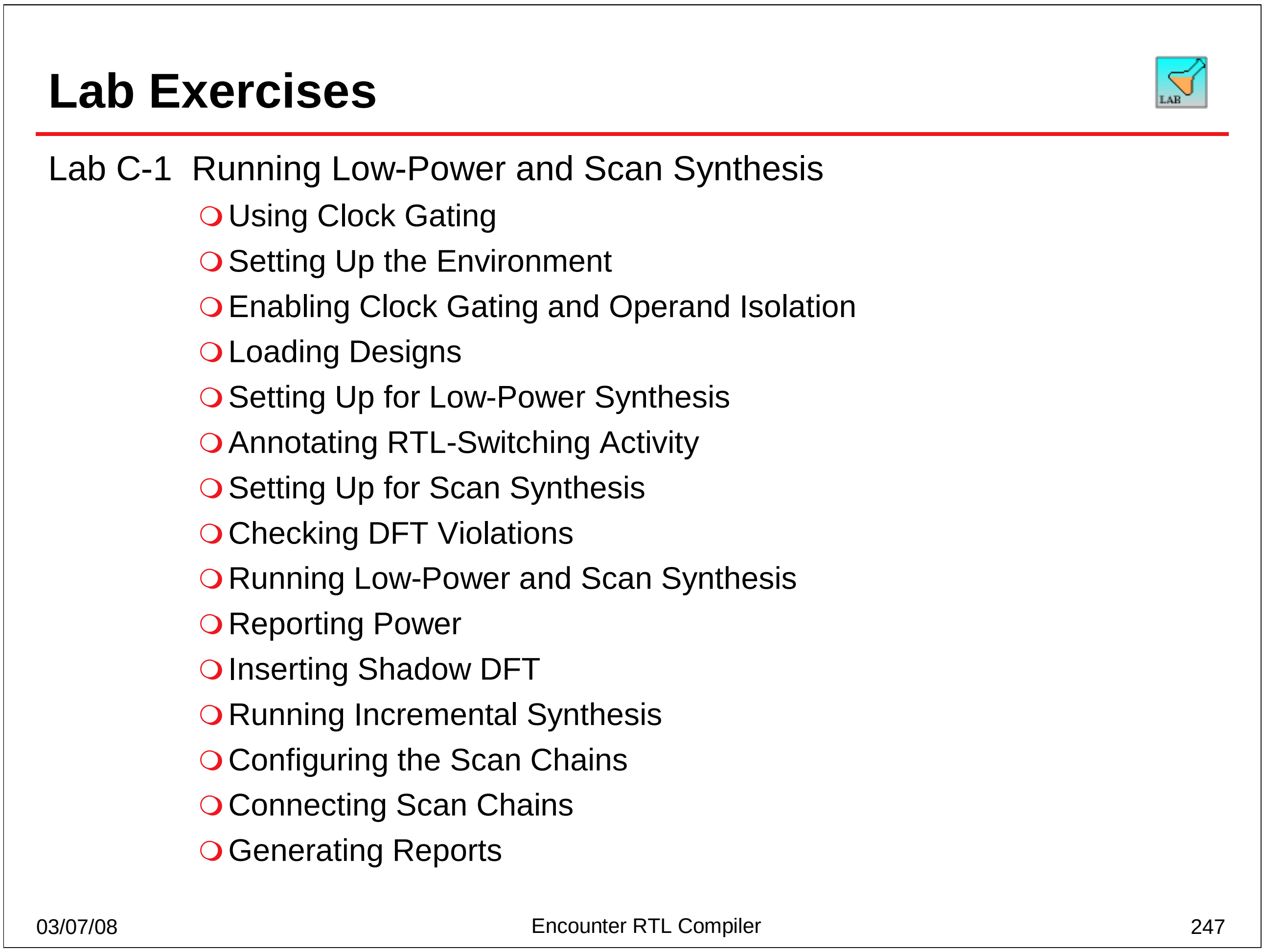
Use low-VT only if timing is not met. Insert clock gating and test logic. Map and optimize for best timing/power/area.

Simulate the design and read the probability data.

Fix remaining DFT violations. Convert remaining testable flops if needed.

Connect the scan chains.

Fix any timing problems due to DFT.



Lab Exercises

Lab C-1 Running Low-Power and Scan Synthesis

- Using Clock Gating
- Setting Up the Environment
- Enabling Clock Gating and Operand Isolation
- Loading Designs
- Setting Up for Low-Power Synthesis
- Annotating RTL-Switching Activity
- Setting Up for Scan Synthesis
- Checking DFT Violations
- Running Low-Power and Scan Synthesis
- Reporting Power
- Inserting Shadow DFT
- Running Incremental Synthesis
- Configuring the Scan Chains
- Connecting Scan Chains
- Generating Reports