

# **Encounter<sup>®</sup> RTL Compiler**

**Version 10.1**

**Lecture Manual**

**April 25, 2011**

© 1990-2011 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

Cadence Design Systems, Inc., 2655 Seely Avenue, San Jose, CA 95134, USA

## Cadence Trademarks

Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address above or call 800.862.4522.

Allegro®	Incisive®	Silicon Express™
Accelerating Mixed Signal Design®	InstallScape™	SKILL®
Assura®	IP Gallery™	SoC Encounter™
BuildGates®	NanoRoute®	SourceLink® online customer support
Cadence®	NC-Verilog®	Specman®
CeltIC®	NeoCell®	Spectre®
Conformal®	NeoCircuit®	Speed Bridge®
Connections®	OpenBook® online documentation library	UltraSim®
Diva®	OrCAD®	Verifault-XL®
Dracula®	Palladium®	Verification Advisor®
ElectronStorm®	Pearl®	Verilog®
Encounter®	PowerSuite®	Virtuoso®
EU CAD®	PSpice®	VoltageStorm®
Fire & Ice®	SignalStorm®	Xtreme®
First Encounter®	Silicon Design Chain™	
HDL-ICE®	Silicon Ensemble®	

## Other Trademarks

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

All other trademarks are the property of their respective holders.

## Confidentiality Notice

No part of this publication may be reproduced in whole or in part by any means (including photocopying or storage in an information storage/retrieval system) or transmitted in any form or by any means without prior written permission from Cadence Design Systems, Inc. (Cadence).

Information in this document is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

UNPUBLISHED This document contains unpublished confidential information and is not to be disclosed or used except as authorized by written contract with Cadence. Rights reserved under the copyright laws of the United States.

## **Table of Contents**

### **Encounter RTL Compiler**

<b>Module 1</b>	<b>About this Course .....</b>	<b>3</b>
<b>Module 2</b>	<b>Introduction to Encounter RTL Compiler .....</b>	<b>19</b>
<b>Module 3</b>	<b>HDL Modeling.....</b>	<b>33</b>
<b>Module 4</b>	<b>Synthesis Flow.....</b>	<b>43</b>
	Lab 4-1 Running the Basic Synthesis Flow	
	Lab 4-2 Navigating the Design Hierarchy	
	Lab 4-3 Reading SDC Constraints	
	Lab 4-4 Using the Graphical Interface	
<b>Module 5</b>	<b>Datapath Synthesis.....</b>	<b>85</b>
	Lab 5-1 Running Datapath Synthesis	
<b>Module 6</b>	<b>Optimization Strategies.....</b>	<b>109</b>
	Lab 6-1 Exploring Optimization Strategies	
<b>Module 7</b>	<b>Low-Power Synthesis .....</b>	<b>143</b>
	Lab 7-1 Running Low-Power Synthesis	
<b>Module 8</b>	<b>Interface to Other Tools .....</b>	<b>191</b>
	Lab 8-1 Interfacing with Other Tools	
<b>Module 9</b>	<b>Test Synthesis .....</b>	<b>207</b>
	Lab 9-1 Running Scan Synthesis	

<b>Appendix A</b>	<b>Physical Synthesis.....</b>	<b>263</b>
<b>Appendix B</b>	<b>Advanced Synthesis Features .....</b>	<b>277</b>
<b>Appendix C</b>	<b>Encounter RTL Compiler Constraints.....</b>	<b>285</b>
	Lab C-1 Applying RTL Compiler Constraints (Optional)	

# Encounter® RTL Compiler

**Version 10.1**



**April 25, 2011**

---

Blank Page

# About This Course

## Module 1



April 25, 2011

# Course Prerequisites

---

Before taking this course, you must have experience with or already have knowledge of the following:

- ◆ Any HDL such as Verilog® (recommended) or VHDL
- ◆ Synthesis and ASIC design flow basics
- ◆ Static timing analysis basics

As an alternative, you can complete the following courses:

- ◆ Basic Static Timing Analysis
- ◆ Verilog Language and Application

ASIC: Application specific integrated chip



# 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)

# Course Modules

## Session 1

- ❑ Introduction to Encounter RTL Compiler
- ❑ HDL Modeling
- ❑ Synthesis Flow

## Session 2

- ❑ Datapath Synthesis
- ❑ Optimization Strategies

## Session 3

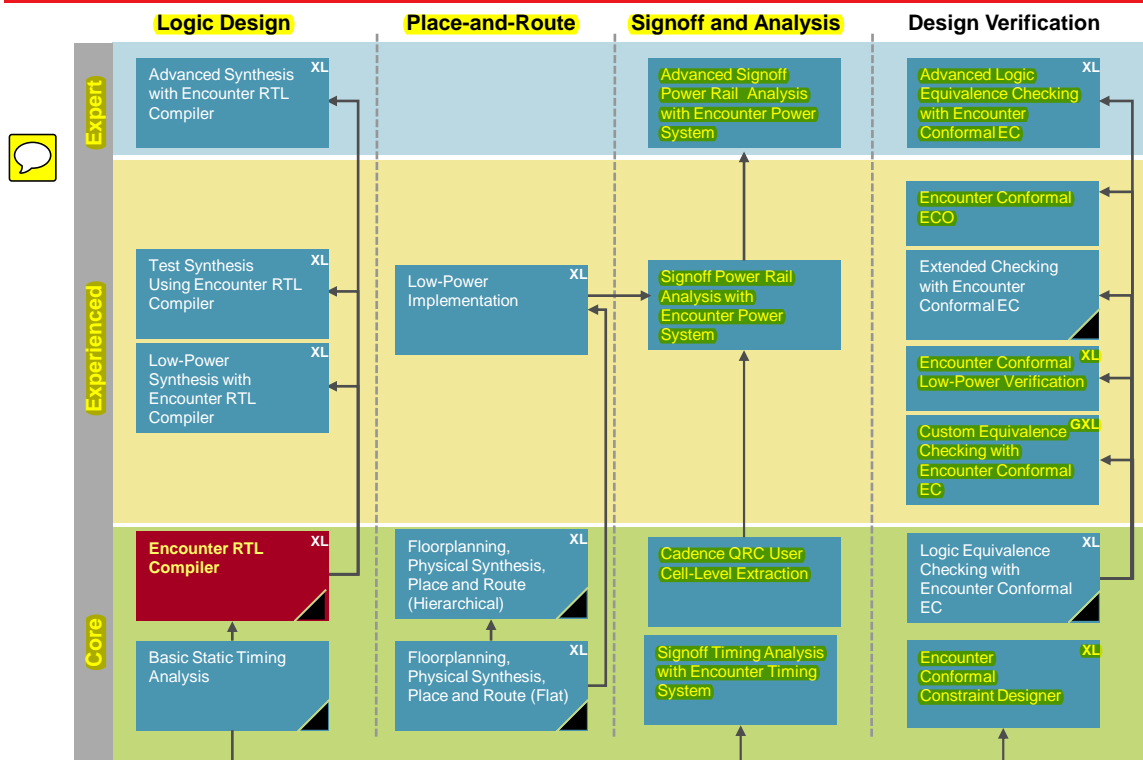
- ❑ Low-Power Synthesis

## Session 4

- ❑ Test Synthesis
- ❑ Interface to Other Tools
- ❑ Appendix A: Advanced Features
- ❑ Appendix B: RTL Compiler Constraints

**Each session typically runs about 3-4 hours**

# Learning Map of Digital Implementation Courses



▲ Also available as an Internet Learning Series course. L, XL, GXL denote tiers of Cadence products.  
Course titles may vary. Please refer to your regional catalog for exact titles and course datasheets.

04/25/11

7

For more information about Cadence® courses:

- Go to [www.cadence.com](http://www.cadence.com).
- Click **Support & Training**.
- Choose one of the catalogs to explore.

## Training Course Catalogs

Courses are available in your region at local training centers.

Browse the course catalogs:

- EMEA
- India
- North America
- China
- Japan
- Korea
- Taiwan

# Getting Help

---

There are three ways to get help:

- ◆ Cadence Help: Accessing tool-specific help
  - ❑ `cdnshelp`
- ◆ Cadence Online Support: Accessing documents and support
  - ❑ <http://support.cadence.com>
- ◆ Cadence Community: Staying current with tips and tricks
  - ❑ <http://www.cadence.com/community/forums>
  - ❑ <http://www.cadence.com/community/blogs>
  - ❑ <http://www.cadence.com/cdnlive>

# Cadence Help

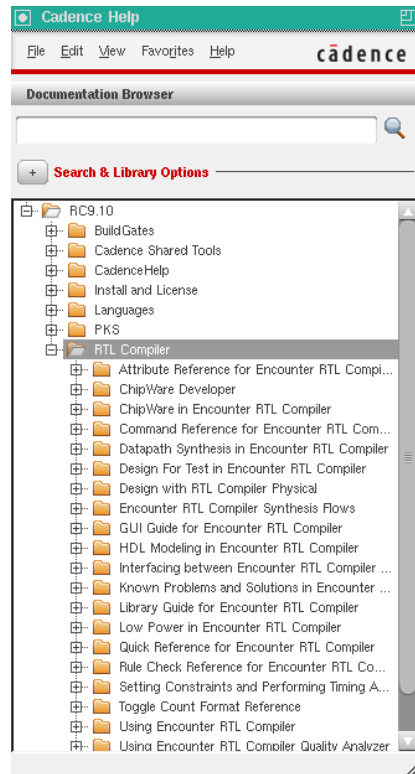
Cadence Help gives you access to the Cadence online product documentation system.

Documentation for each product is included automatically when you install the product. Documents are available in both HTML and PDF format.

The Library window lets you access documents by product family, product name, or type of document.

You can access Cadence Help from

- ◆ The graphical user interface
  - ❑ The Help menu in windows
  - ❑ The Help button on forms
- ◆ The command line: *cdnshelp*
- ◆ The Cadence Online Support system (if you have a license agreement)



04/25/11

Encounter RTL Compiler

9

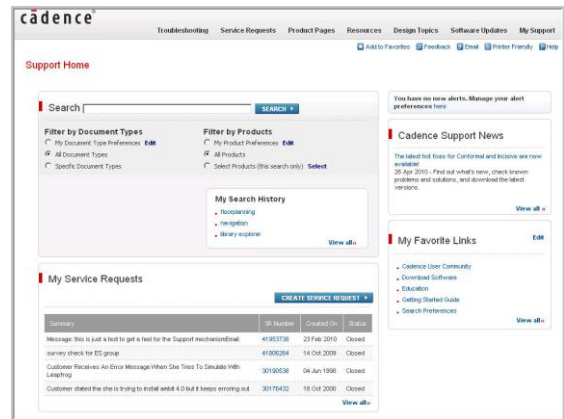
- To access Cadence Help from a Cadence software application, from the menu, choose **Help—Cadence Documentation**. The browser displays the document page. After the document page opens, click the **Library** button to open the Library window.
- To access Cadence Help from a command line, enter *cdnshelp &* in a terminal window. When the Library window appears, navigate to the manual you want by selecting the specific product, then double-click to open the manual in your browser.

# Cadence Online Support (COS)

Cadence Online Support (<http://support.cadence.com>) is a website that gives you access to support resources, including

- ◆ An extensive knowledge base with
  - ❑ User guides
  - ❑ Reference manuals
  - ❑ Design topics
  - ❑ Frequently asked questions
  - ❑ Known problems and solutions
  - ❑ White papers
  - ❑ Application notes
- ◆ Software updates for Cadence products
- ◆ Access to Cadence customer support engineers

Register now and take advantage of the many benefits of Cadence online support.



04/25/11

Encounter RTL Compiler

10

## Searching the Knowledgebase:

You can personalize the document types and products by editing your Preferences.

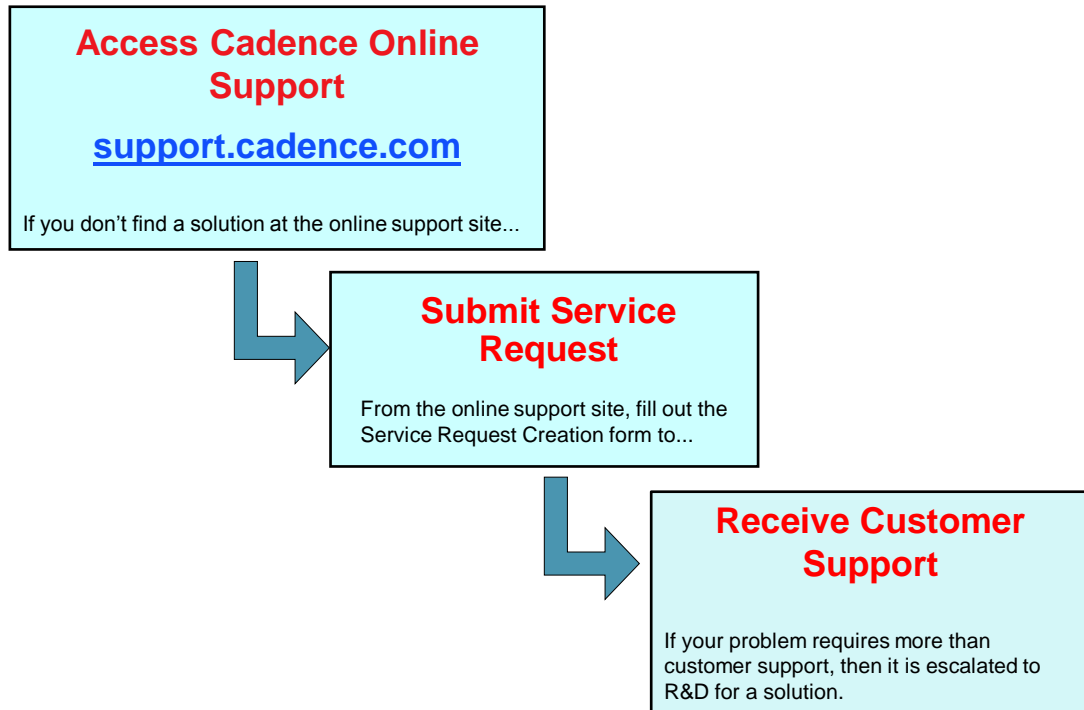
To search the knowledge base for different document types:

1. Point your web browser to [support.cadence.com](http://support.cadence.com).
2. Log in to the online support site.
3. Enter search criteria.
4. Specify document type for current search
5. Select Products for this search only
6. Click Resources menu to access:
  - Application Notes,
  - Computing Platforms,
  - Installation Information,
  - Product & Release Lifecycle
  - Product Manuals
  - SKILL Information
  - Tech Info and White Papers
  - Links to Other Resources
  - Troubleshooting Information
  - Video Library

Once search results are displayed, you can use the “within” option to refine your search. You can also filter your results based on the year created, “Date Filter”.

# Using Cadence Online Support

---



04/25/11

Encounter RTL Compiler

11

You can use the Cadence Online Support site for service requests. Fill out the Service Request Creation form and submit it. The request goes first to Customer Support, and if necessary is escalated to Cadence R&D for a solution.

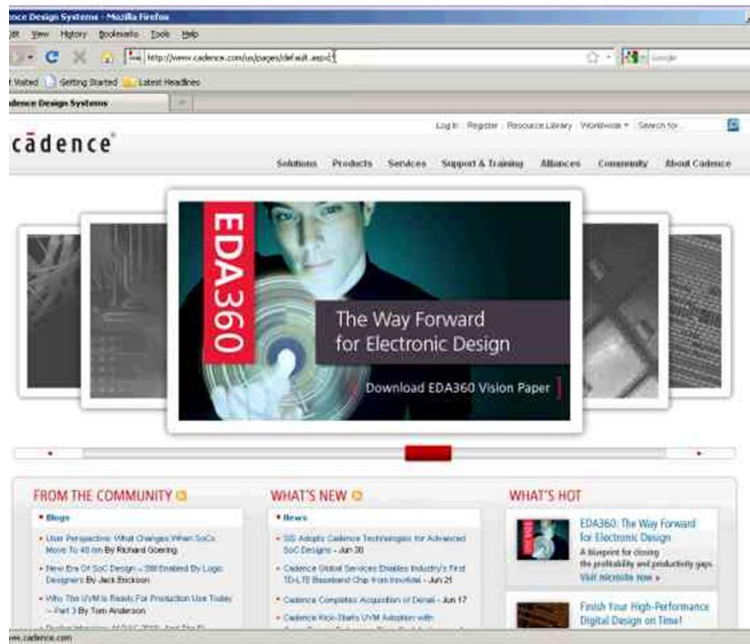
## Submitting a Service Request:

1. Point your web browser to [support.cadence.com](http://support.cadence.com).
2. Log in to the online support site.
3. Click on "Create Service Request" to interact directly with Cadence Customer Support
4. Select the product from the list of products
5. Describe the problem
6. Click Continue
7. View documents that may solve your stated problem
8. Set additional request attributes
9. Click Submit SR

# Demo: Cadence Online Support

The demo at right shows you how to:

- ◆ Register for Cadence Online Support
- ◆ Search the knowledge base
- ◆ Submit a service request
- ◆ View your service request



<http://support.cadence.com>

04/25/11

Encounter RTL Compiler

12

This section outlines how to register for COS.

## Registering for Cadence Online Support

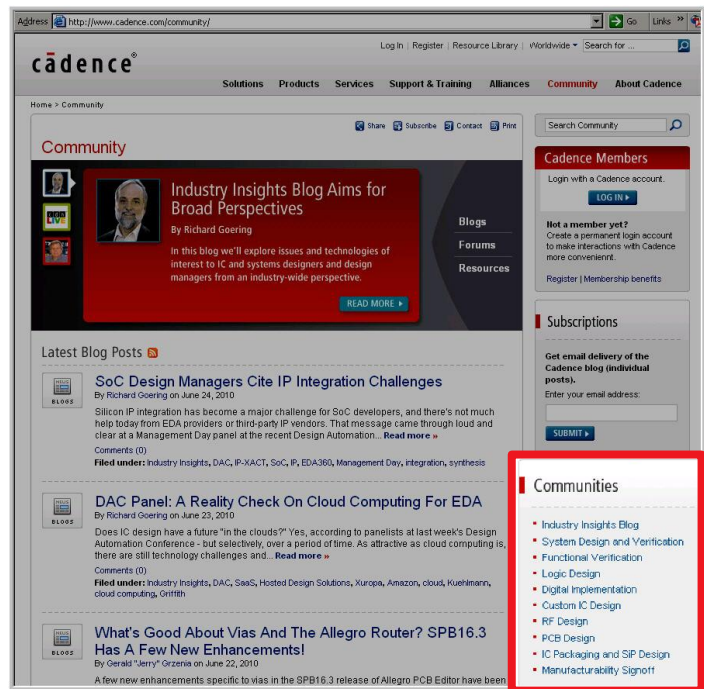
1. Go to <http://support.cadence.com>
2. Click the “Register Now” link
3. You will need:
  - Email address you use at work
  - You may also be asked for your license server Host Id or Reference Key
4. After registering, check your email.
5. Use your email ID and the one time password provided in the email, to login
6. You will then change your password, security question, and personal data



# Cadence Online Communities

Stay connected by visiting resources, such as blogs and forums,

1. Go to:  
<http://www.cadence.com/community>
2. Select your area of interest.



# Lab Exercises

---



## Lab 1-1 Locating Cadence Online Support Solutions

- ◆ Log in and run a search.

## Lab 1-2 Customizing Notification and Search Preferences

- ◆ Set preferences to improve search and receive email notification.

# Lab Instructions

---

You can only complete these labs if you have access to the internet and a Cadence Online Support account. If you do not, your instructor may be able to perform a demo of these labs for the class.

**Lab 1:** Log in to Cadence Online Support (COS) and search for information about a specific issue.

1. In a web browser enter  
`http://support.cadence.com`
2. Log in to Cadence Online Support with your email and password
3. In the Support Home page, make sure the following options are selected:
  - ◆ All Document Types
  - ◆ All Products
4. In the Search field enter  
<search term to be provided by instructor>  
and click the **SEARCH** button.
5. In the Troubleshooting Info section, click the match called:  
<item to select provided by instructor>
6. Close the solution window.

**Lab 2:** Set preferences so you can improve search results and receive email notification.

1. Click on the **My Account** link.
2. Click on the **Notification Preferences** tab.
3. On the Set Email Notification Preferences page:
  - a. Check **Send me email notifications about new product releases**.
  - b. Click on the products, email format and document types you prefer.
  - c. Click **Save**.
4. Click on the **Search Preferences** tab.
5. On the **Set Search Preferences** page:
  - a. Click on **Use same product and document type preferences as my Notification Preferences**.
  - b. Click **Save**.

**Note:** You can filter the search results by selecting specific document types or products which are listed to the left of the results.

# Certificate of Course Completion (Optional)

---

- ◆ Benefits:
  - ❑ Test your knowledge and understanding.
  - ❑ An acknowledgement of course mastery.
  - ❑ Update your resume with new skills and make yourself more marketable.
  - ❑ Peer recognition.
- ◆ To receive a Certificate of Completion, you must complete an online exam with a score of 75% or more.
- ◆ Detailed instructions for obtaining a certificate of completion are provided at the end of this course.

You can obtain a proof of attendance after attending a Cadence Instructor-led Course (Public, Standard onsite, or Virtual Class). Please contact your local regional customer training office\* to request a Certificate of Attendance. Be sure to include your name and course title.

\* For local customer training contact information, refer to:  
[http://www.cadence.com/Training/Pages/training\\_contacts.aspx](http://www.cadence.com/Training/Pages/training_contacts.aspx)

# Pre and Post Assessments

---

Pre and Post Assessments are a way to help you determine what new knowledge you have gained by completing this course.

- ◆ You will take a *Pre-class* Assessment at the start of the class.
- ◆ You will take a *Post-class* Assessment at the end of the class.
- ◆ These will have the same set of questions and take about **15 minutes** to complete.
- ◆ After the *Post-Assessment* we will go over the questions and answers in detail.

The expectation is that you will not successfully answer most of the questions in the pre class assessment, which is expected as you have not yet completed the class.

Conversely, at the end of the class you should be able to answer correctly most of the post assessment questions.

# Completing the Pre-Class Assessment

---

1. In a web browser enter: <http://exam.cadence.com>
2. Login to the exam server:
  - a) **Name:** your complete email address (example: [joe@cadence.com](mailto:joe@cadence.com))
  - b) **Group:** your company's email suffix (example: [cadence.com](#))
3. Select the Pre-class assessment for this class:  
ES <your course title> **PRE**
4. Complete the assessment. You do not need to answer all questions.
5. Click Submit at the bottom of the exam. *Note: You will be given a score but will not be provided with the answers until you take the post assessment at the end of class.*

# Introduction to Encounter RTL Compiler

## Module 2



April 25, 2011

# Module Objectives

---

In this module, you

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



# What Is Logic Synthesis?

## Definition:

The process of parsing, translating, optimizing, and mapping RTL code into a specified standard cell library.

To determine the feasibility of the design, we need to synthesize the RTL code into gates and measure timing, power, and area, before moving on to placement.

```
...  
assign #1 arnz = (ar != 0) ;  
...
```

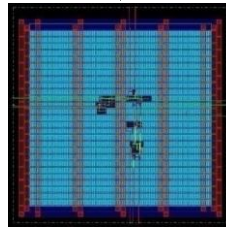
**RTL Code**

```
NAND4X1MTH g395 (.A (n_5), .B (n_7),  
.C (n_10), .D (n_12), .Y (arnz) );
```

**Mapping  
RTL to  
Library**

Timing, Power, Area  
**Ok?**

yes



**Placed and  
Routed  
Design**

04/25/11

Encounter RTL Compiler

# Examples of Logic Synthesis

RTL	Generic Netlist	Synthesized Netlist
<pre>assign #1 gez = gz   z;</pre>	<pre>or g1 (gez, gz, z);</pre>	<pre>assign gez = 1'b0;  // Zeroed because // of optimization</pre>
<pre>assign #1 arnz = (ar != 0) ;</pre>	<pre>nor g45 (n_63, n_61, n_62); not g46 (arnz, n_63);  // Not targeted to // a library</pre>	<pre>NAND4X1MTH g395(.A (n_5), .B (n_7), .C (n_10), .D (n_12), .Y (arnz));  // Notice that the // gate is mapped to a // library component.</pre>

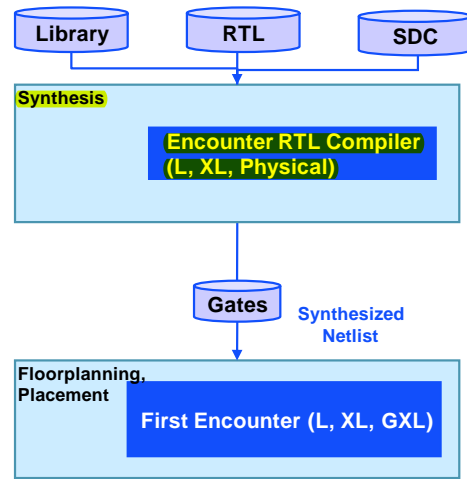
# Logic Synthesis: Input and Output Formats

## ◆ Input

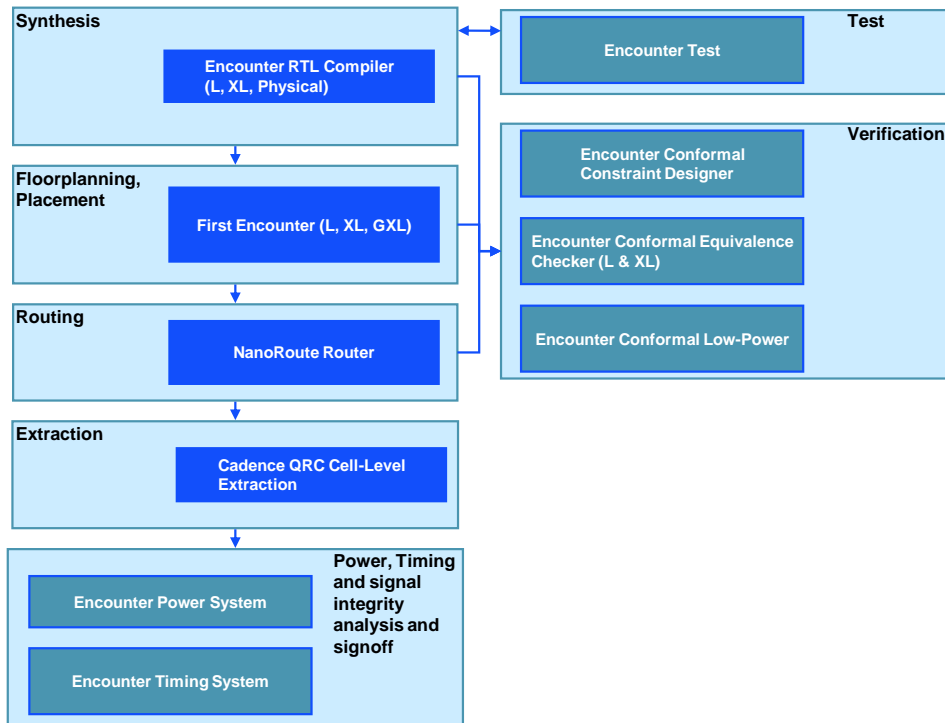
- ❑ RTL in the Verilog® language or other HDL
- ❑ Constraints in Synopsys Design Constraints (SDC) format
- ❑ Timing libraries in Liberty (.lib) format

## ◆ Output

- ❑ Gate level netlist in the Verilog language



# Cadence Encounter Technology



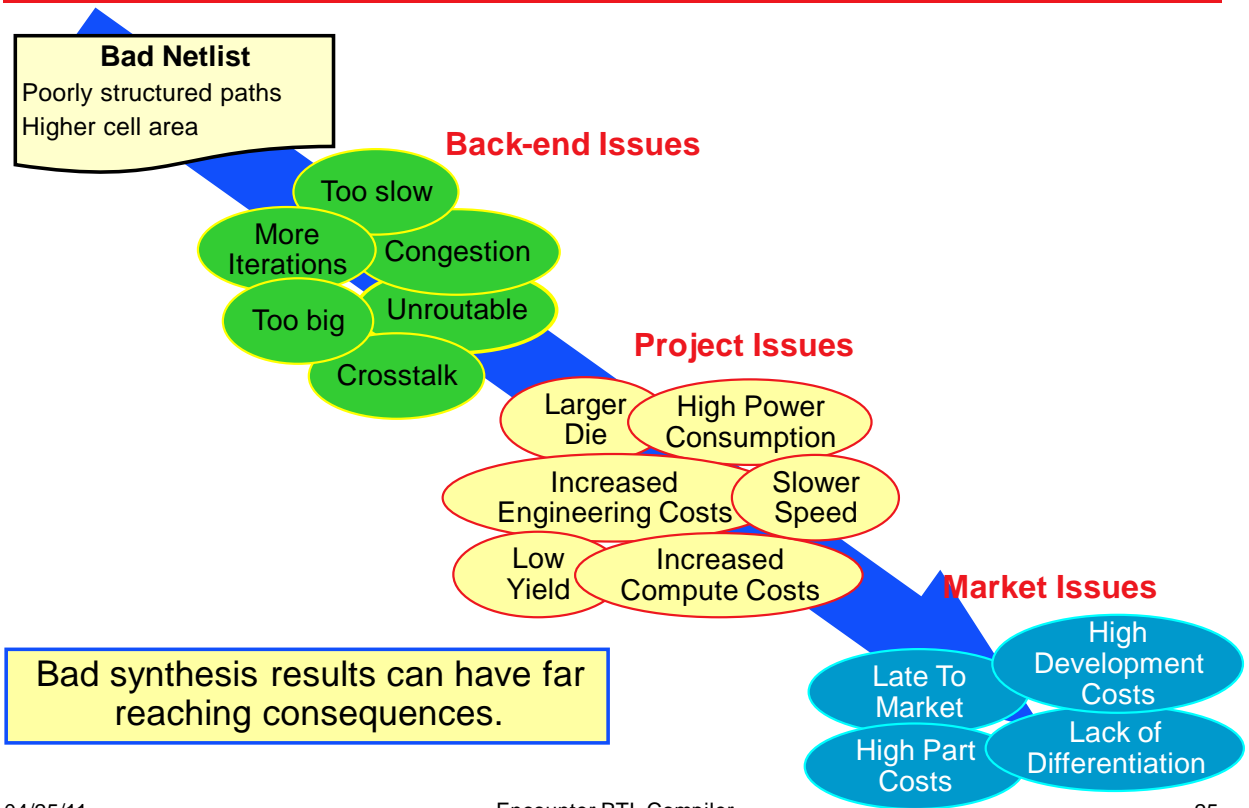
04/25/11

Encounter RTL Compiler

24

Simulation and layout tools that are adjacent to the Encounter technology are not shown so that the graph is concise.

# Does RTL Synthesis Really Matter?

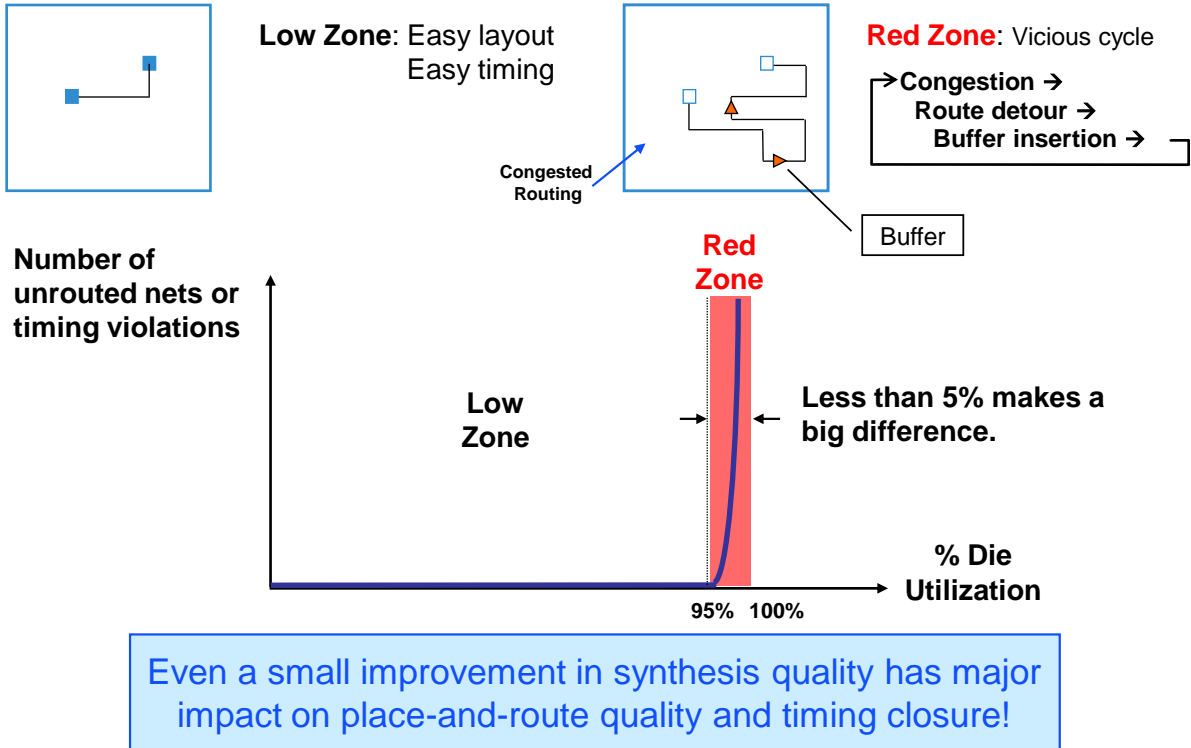


04/25/11

Encounter RTL Compiler

25

# Synthesis Impact on Place and Route



04/25/11

Encounter RTL Compiler

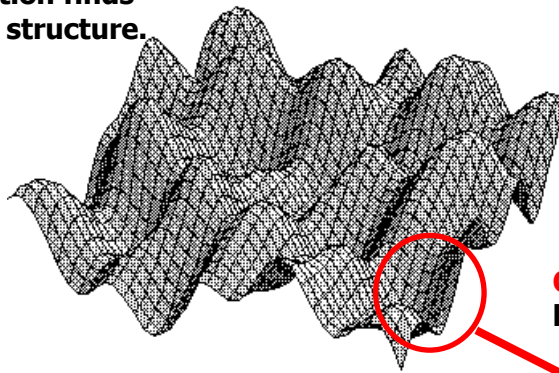
26

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.

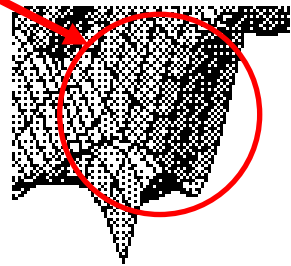
# Importance of Global Optimization

**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!



04/25/11

Encounter RTL Compiler

27

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.

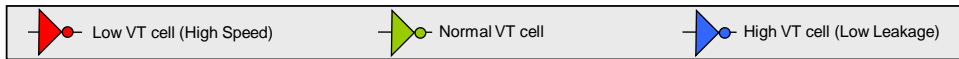
In contrast, Encounter® RTL Compiler algorithms iterate through multiple design scenarios that provide the best initial design to create a solution that is far superior to those achievable using the other synthesis tools.

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.

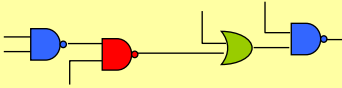
As designs get larger, the benefits of a global optimization strategy become increasingly important.

# Better Structure for Place and Route

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



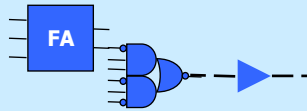
## Critical Path Mapping



- 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**

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.

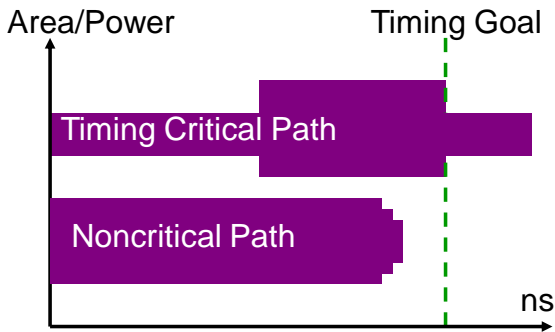
Noncritical paths are structured to be smaller and to cause less leakage. Encounter RTL Compiler will optimize for leakage even with a single VT library.

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



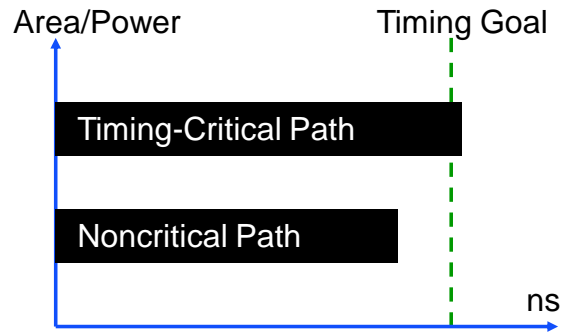
# How Is the Global Approach Different?

## 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.

04/25/11

Encounter RTL Compiler

29

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

Low capacity → small blocks

Peep hole on path → local minima

Path at a time → nonlinear run time

Axis at a time → suboptimal QoS

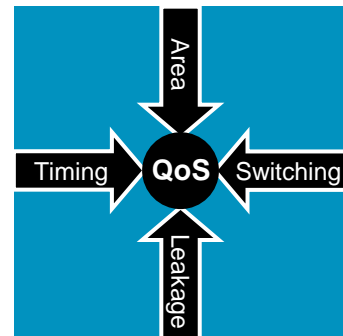
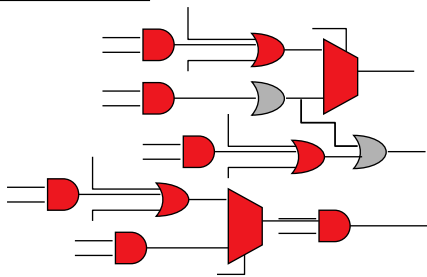
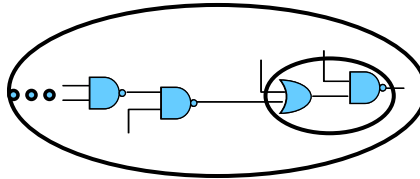
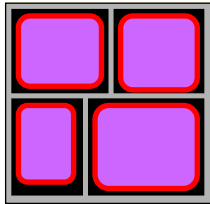
## NEW ⇒ ↑QoS

High capacity → global solution view

Whole path → balanced path structure

Multi-path → interrelationships leveraged

Whole objective space → single pass



04/25/11

Encounter RTL Compiler

30

## QoS: Quality of Silicon

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.

# RTL Compiler Product Packaging

Global synthesis engine				
Multi-vt optimization				
Clock gating	VDI	RC-L		
Datapath synthesis	3002	(RC110)		
ChipWare				
DFT rule checking and fixing	Capacity – limited	Capacity – limited	RC-XL	
Scan insertion	50k instances Can stack 2	100k mapped/ 400k generic	(RC200)	RC-Physical
RTL/gate power estimation				(RC400)
Power shutoff (PSO) synthesis	Comes with capacity-limited	4 stack = Unlimited capacity		
PLE & RC-Spatial coarse physical prediction	EDI System			
Superthreading master				
Cache-based Rapid Re-synthesis				
RC-Physical				
• Legal placement				
• Physical optimization				
• Congestion optimization				
Top-down retiming	RC-GXL option			
Top-down multi-supply voltage (MSV)				
Single-pass multi-mode synthesis	(RC300)			
Dynamic voltage-frequency scaling	Requires RC-L, RC-XL, or equivalent as base			

04/25/11

Encounter RTL Compiler

31

RC-Physical includes a superthreading master along with an extra thread license (total=2 threads).

RC-XL is available via 1P1L in the following, to which an RC-GXL can be paired:

- RC-Verify (RCV200)
- RC-Physical (RC400)
- Conformal ECO (CFM620)
- C-to-Silicon (CTS102)
- FE-XL (FE100GPS)
- SoCE-XL (FE200GPS)

RC-XL+RC-GXL available via 1P1L in:

- FE-GXL (FE800)
- SoCE-GXL (FE850)

RC provides BG access as 1P1L

- BG with RC-L
- BGX with RC-XL

\*The “generic” limit is because we don’t know the final instance count until the end of the synthesis run, and users wanted the job to terminate earlier if it didn’t fit in the capacity of the license. So we do a check of generic instance count at each step in the optimization, and the limit here is 4x the mapped count.

SuperThreading master is available in all RC packages.

---

Blank Page

# HDL Modeling

## Module 3



April 25, 2011

33

# Module Objectives

---

In this module, you

- ◆ Identify a few HDL modeling techniques
- ◆ Identify where pragmas can be used

# Modeling a Rising-Edge-Triggered Flip-Flop

RTL	Synthesized Netlist
<pre>module sync_flop (clk, din, dout);     input clk;     input din;     output dout;     reg dout;     always @(posedge clk)         begin             dout &lt;= din;         end endmodule</pre>	<pre>module sync_flop(clk, din, dout);     input clk, din;     output dout;     wire clk, din;     wire dout;     DFFQX2M dout_reg(.CK (clk), .D (din), .Q (dout)); endmodule</pre>

# Modeling Clock Gating Logic

RTL	Synthesized Netlist
<pre>module conv_subreg(rcc_clk,   enable, din, dout) ; ... always @(negedge rcc_clk)   dout = enable ? din : dout; endmodule</pre>	<pre>module conv_subreg(rcc_clk,   enable, din, dout) ; ... TLATNTSCAX2M RC_CGIC_INST(.E(enable), .CK(ck_in), .SE (test), .ECK(rc_clk));  DFFQX2M \dout_reg[1] (.CK (rcc_clk), .D (din[1]), .Q (dout[1])); ... DFFQX2M \dout_reg[7] (.CK (rcc_clk), .D (din[7]), .Q (dout[7])); ... endmodule</pre>

04/25/11

Encounter RTL Compiler

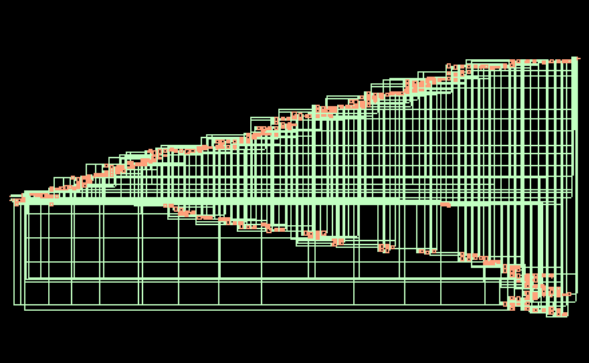
36



# Modeling Adders

RTL	Synthesized Netlist
<pre>module signed_add (Y, A, B);   parameter w = 16;   input [w-1:0] A, B;   output [w-1:0] Y;   input signed A, B;   output signed Y;   assign Y = A + B; endmodule</pre>	<pre>module signed_add(Y, A, B);   input [15:0] A, B;   output [15:0] Y;   wire [15:0] A, B;   wire [15:0] Y;   ...   ADDFX2MTH g379(.A (B[2]), .B (A[2]), .CI (n_3), .S (Z[2]), .CO (n_5));   ADDFX2MTH g380(.A (B[1]), .B (A[1]), .CI (n_1), .S (Z[1]), .CO (n_3));   ... endmodule</pre>

# Modeling Division

RTL	Synthesized Netlist
<pre>module unsigned_divide (Y, A, B);   parameter wA = 16, wB = 6;   input [wA-1:0] A;   input [wB-1:0] B;   output [wA-1:0] Y;   assign y = A / B; endmodule</pre>	<pre>module unsigned_divide(A, B, Y);   input [15:0] A;   input [5:0] B;   output [15:0] Y;   ... </pre>

# Modeling Case Statements

---

When a case statement does not specify all possible case condition values, a latch is inferred. If RTL Compiler determines that the case is not full, then it uses a latch to implement a state transition table.

## Case statement to infer a Latch

```
module case_latch(dout,sel,a,b,c);
    input [1:0] sel;
    input a,b,c;
    output dout;
    reg dout;
    always @(a or b or c or sel)
    begin
        case (sel)
            2'b00 : dout = a;
            2'b01 : dout = b;
            2'b10 : dout = c;

        endcase
    end
endmodule
```

## Case statement to **NOT** infer a Latch

```
module case_nolatch(dout,sel,a,b,c);
    input [1:0] sel;
    input a,b,c;
    output dout;
    reg dout;
    always @(a or b or c or sel)
    begin
        case (sel)
            2'b00 : dout = a;
            2'b01 : dout = b;
            2'b10 : dout = c;
            default : dout = 1'b0;

        endcase
    end
endmodule
```

# Synthesis Pragmas

---

Synthesis pragmas are specially-formatted comments. Do not confuse these comments with Verilog HDL compiler directives that begin with ```. Synthesis pragmas perform code selection or specify how the set and reset pins of a register are wired.

RTL Compiler supports the following two forms of Verilog synthesis pragmas:

- ◆ Short comments that terminate at the end of the line:

```
// cadence pragma_name
```

- ◆ Long comments that extend beyond one line:

```
/* cadence pragma_name */
```

# Specifying the `parallel_case` Pragma

Use the *parallel\_case* pragma to specify that the condition values listed for the case statement in the RTL are non-overlapping. This can result in a more efficient design because it prevents RTL Compiler from generating logic to prioritize the conditions.

```
module case_parallel
  (clk,cc,data_in,cntr);
  input clk;
  input [3:0] cc;
  input [2:0] data_in;
  output [2:0] cntr;
  reg [2:0] cntr;
  always @(posedge clk) begin
    case (1'b1) // cadence parallel_case
      cc[0] : cntr = 0 ;
      cc[1] : cntr = data_in ;
      cc[2] : cntr = data_in - 1 ;
      cc[3] : cntr = data_in + 1 ;
    endcase
  end
endmodule
```

The `parallel_case` pragma directs RTL Compiler to implement the following logic for `cntr`.

```
cntr = (cc[0] and '0') or
(cc[1] and data_in) or
(cc[2] and data_in-1) or
(cc[3] and data_in+1)
```

rather than:

```
cntr = (cc[0] and '0') or
(!cc[0] and cc[1] and data_in) or
(!cc[0] and !cc[1] and cc[2] and
data_in-1) or
(!cc[0] and !cc[1] and !cc[2] and cc[3]
and data_in+1)
```

Comparing the RTL to the synthesized design using formal verification or simulation may result in mismatches if overlapping case condition values, such as `cc = 1111` shown in the example, are applied during verification to case statements marked as *parallel\_case*.

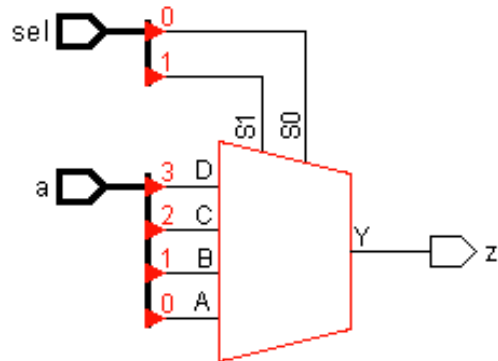
# Specifying Multiplexer Mapping Pragma

Use the *map\_to\_mux* pragma (also called *infer\_mux*) in the *case* and *if-then-else* statements to force RTL Compiler to implement the statement with multiplexer components from the technology library.

## RTL

```
module map2mux1 (a,sel,z);
  input [3:0] a;
  input [1:0] sel;
  output z;
  reg z;
  always @ (a or sel) begin
    case (sel) // cadence map_to_mux
      2'b00 : z <= a[0];
      2'b01 : z <= a[1];
      2'b10 : z <= a[2];
      2'b11 : z <= a[3];
    endcase
  end
endmodule
```

The *map\_to\_mux* pragma directs RTL Compiler to implement a MUX for the case statement.



The resulting netlist may have worse area, delay, or power than if RTL Compiler were not forced to map to multiplexers.

# Synthesis Flow

## Module 4



April 25, 2011

# 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



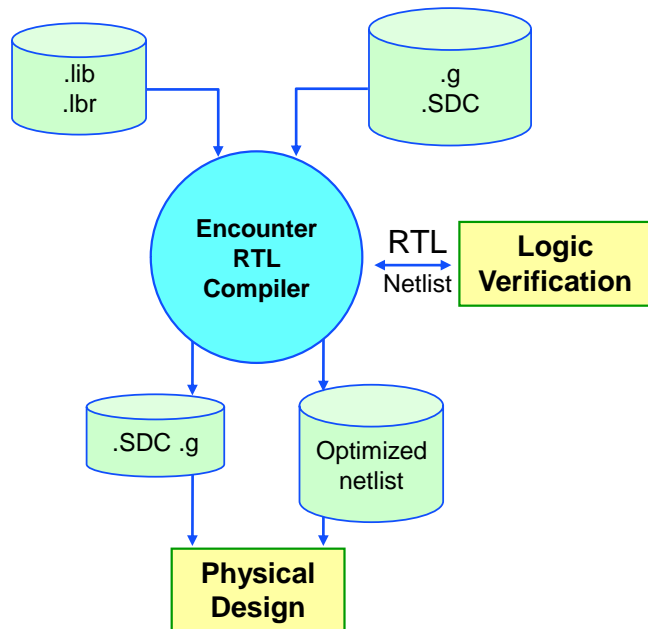
# Inputs and Outputs of Encounter RTL Compiler

## 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



04/25/11

Encounter RTL Compiler

45

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

# 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]  
  [-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.)
- [-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 superthreading
- [-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

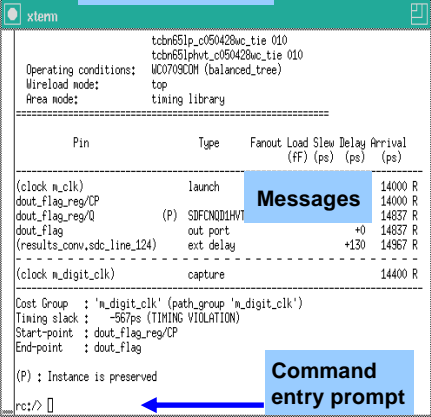
# Graphical User Interface (GUI) Environment

Operating system interface window

Read design info

Timing and other reports

Debug tools



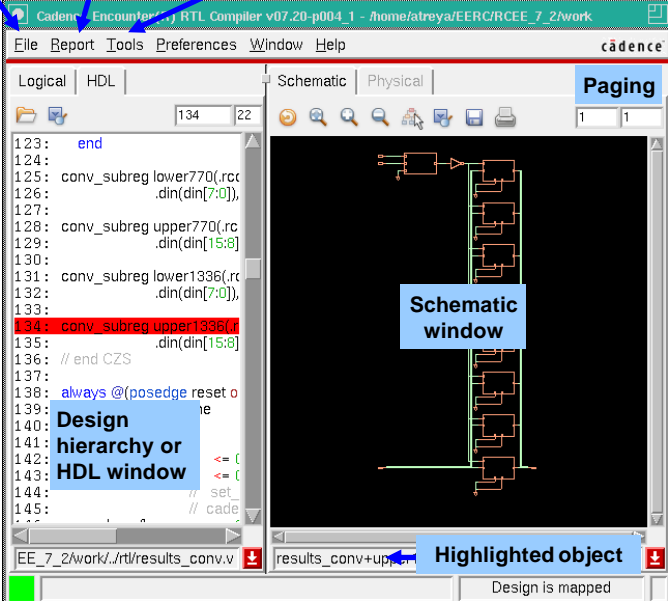
**Operating conditions:**  
Wireload mode: top  
Area node: timing library

Pin	Type	Fanout	Load	Slew	Delay	Arrival
		(ff)	(ps)	(ps)	(ps)	(ps)
(clock_m_clk)	launch					14000 R
dout_flag_reg/CP						14000 R
dout_flag_reg/Q	(P) SDFCND01HVT					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:/> Command entry prompt



**Design hierarchy or HDL window**

```

123: end
124:
125: conv_subreg lower770(.rc
126:   .din(din[7:0]),
127:
128: conv_subreg upper770(.rc
129:   .din(din[15:8]
130:
131: conv_subreg lower1336(.rc
132:   .din(din[7:0]),
133:
134: conv_subreg upper1336(.rc
135:   .din(din[15:8]
136: // end CZS
137:
138: always @(posedge reset o
139:   ne
140:
141: <= (
142: <= (
143: <= (
144: // set_
145: // cade

```

**Schematic window**

**Highlighted object**

**Status**

Design is mapped

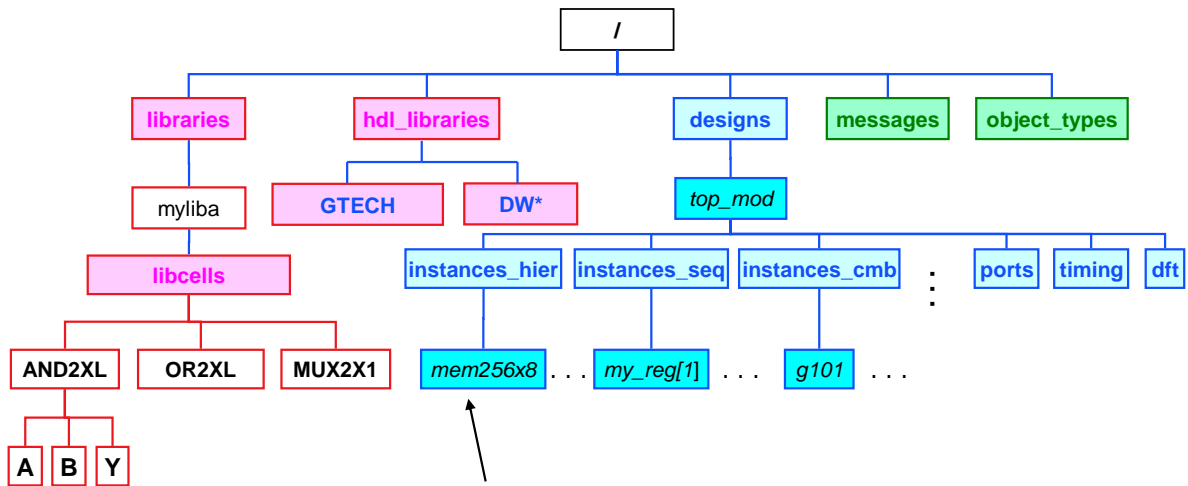
Use gui\_show gui\_hide commands to show or hide the graphical interface.

04/25/11

Encounter RTL Compiler

47

# 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
```

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_attribute 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_attr -h *clock* clock
```

04/25/11

Encounter RTL Compiler

50

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
```

# 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
```

04/25/11

Encounter RTL Compiler

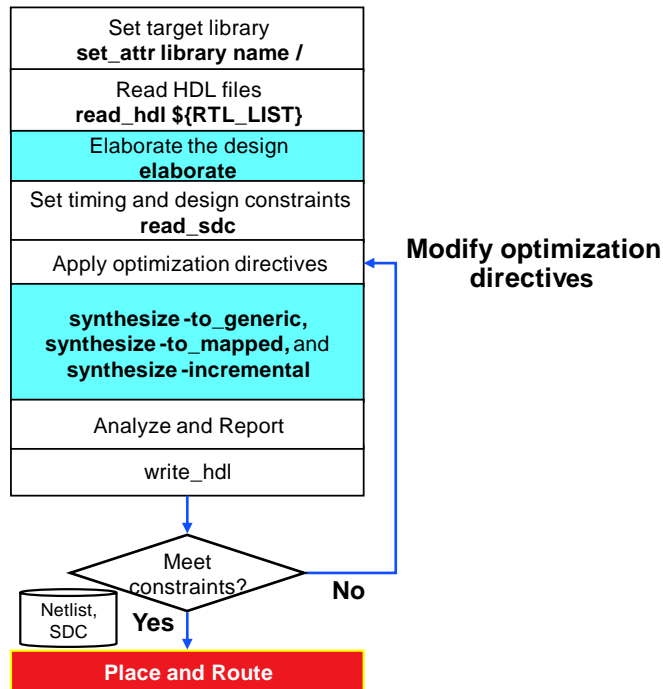
51

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.

# Basic Design Flow

---





# 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

# Generating a Template Script (continued)

---

## Example

```
rc> write_template -split -outfile run.tcl
```

This will create two script files. Use these to operate RC:

*setup\_run.tcl & run.tcl*

- ◆ RC is a true Tcl based tool, it's run scripts are all Tcl scripts.
- ◆ They utilize variables, lists, objects, attributes, directories and commands.
- ◆ Look for the angle brackets <name> to find what you need to provide.

04/25/11

Encounter RTL Compiler

54

Before you proceed to edit the run scripts...

- Use the RC supporting documents. They will be useful to explain new & unfamiliar commands, objects, attributes, and variables used in the template scripts.
- There are more than 17 documents that come with RC.

You can find them under the directory where the tool was installed on your system:

```
unix> which rc
```

```
$path_to_tool_directory/bin/rc
```

```
unix> acread $path_to_tool_directory/doc/rc_XXX/rc_XXX.pdf
```

Most Useful RC Documents:

- rc\_user.pdf : RC User Guide
- rc\_attref.pdf : RC Attribute Reference Guide
- rc\_comref.pdf : RC Command Reference Guide
- rc\_dft\_ug.pdf : RC DFT User's Guide
- rc\_lps\_ug.pdf : RC Low Power User's Guide
- rc\_phys.pdf : RC Physical User's Guide

# Edit setup\_run.tcl

```
#####
#      MAIN SETUP (root attributes & setup variables)      #
#####
## Preset global variables and attributes
#####
set DESIGN    top
set SYN_EFF medium
set MAP_EFF medium
set DATE [clock format [clock seconds]format "%b%d-%T"]
set _OUTPUTS_PATH outputs_$(DATE)
set _REPORTS_PATH reports_$(DATE)
set _LOG_PATH logs_$(DATE)
##set ET_WORKDIR <ET work directory>
✓ set_attributelib_search_path{. ./lib} /
  set_attributescrypt_search_path{.} /
✓ set_attributehdl_search_path{. ./rtl} /
##Uncomment and specify machine names to enable supethreading.
##set_attributesuper_thread_servers{<machine names>} /

##Defaultundriven/unconnected setting is 'none'.
##set_attributehdl_unconnected_input_port_value0 | 1 | x | none /
##set_attributehdl_undriven_output_port_value 0 | 1 | x | none /
##set_attributehdl_undriven_signal_value      0 | 1 | x | none /

##set_attributewireload_mode<value> /
set_attributeinformation_level7 /
#####
## Library setup
#####
set_attributelibrary{a.lib b.lib c.lib} /

set_attributef_library{abc_tech.lef a.lef b.lef c.lef}/
set_attributecap_table_fileabc.captbl/

##generates <signal>_reg[<bit_width>] format
#set_attributehdl_array_naming_style%s[%d] /
## Turn on TNS, affects global andncr opto
set_attributetns_opto true /
```

04/25/11

Encounter RTL Compiler

55

# Edit run.tcl

```
#### Template Script for RTL->Gate-Level Flow (generated from RC RC9.1.200- v09.10-s201_1)
if {[file exists /proc/cpuinfo]} {
  sh grep "model name" /proc/cpuinfo
  sh grep "cpu MHz" /proc/cpuinfo
}
puts "Hostname : [info hostname]"
#####
## Load Design
#####
#### Including setup file (root attributes & setup variables).
include setup run.tcl
read_hdl-v2001 {low.v mid.v top.v}
elaborate $DESIGN
puts "Runtime & Memory after read_hdl"
timestat Elaboration
check_design-unresolved
#####
## Constraints Setup
#####
read_sdc top.sdc
puts "The number of exceptions is [length [find /designs/$DESIGN-exception *]]"
#set_attribute force_wireload <wireload name> "/designs/$DESIGN"

if {[file exists ${_LOG_PATH}]} {
  file mkdir ${_LOG_PATH}
  puts "Creating directory ${_LOG_PATH}"
}
if {[file exists ${_OUTPUTS_PATH}]} {
  file mkdir ${_OUTPUTS_PATH}
  puts "Creating directory ${_OUTPUTS_PATH}"
}
if {[file exists ${_REPORTS_PATH}]} {
  file mkdir ${_REPORTS_PATH}
  puts "Creating directory ${_REPORTS_PATH}"
}
}
report timing-lint
```

04/25/11

...

Encounter RTL Compiler

56

# More run.tcl: No Further Editing Required!

```
#####
## Define cost groups (clockclock, clock-output, input-clock, input-output)
#####
## Uncomment to remove already existing cost groups before creating new ones.
## rm [find /designs/*-cost_group*]
if {[length [all::all_seqs]] > 0} {
  define_cost_group-name I2C -design $DESIGN
  define_cost_group-name C2O -design $DESIGN
  define_cost_group-name C2C -design $DESIGN
  path_group-from [all::all_seqs] -to [all::all_seqs] -group C2C -name C2C
  path_group-from [all::all_seqs] -to [all::all_outs] -group C2O -name C2O
  path_group-from [all::all_inps] -to [all::all_seqs] -group I2C -name I2C
}
define_cost_group-name I2O -design $DESIGN
path_group-from [all::all_inps] -to [all::all_outs] -group I2O -name I2O
foreach cg [find /-cost_group*] {
  report timing-cost_group[list $cg] >> $_REPORTS_PATH/${DESIGN}_prelim.rpt
}

#### To turn off sequential merging on the design
#### uncomment & use the following attributes.
##set_attributeoptimize_merge_flopsfalse /
##set_attributeoptimize_merge_latchesfalse /
#### For a particular instance use attributeoptimize_merge_seq$ to turn off sequential merging.

#####
## Synthesizing to generic
#####
synthesize-to_generic-eff $SYN_EFF
puts "Runtime & Memory after 'synthesizeto_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_generic.rpt

#####
## Synthesizing to gates
#####
synthesize-to_mapped-eff $MAP_EFF-no_incr
puts "Runtime & Memory after 'synthesizeto_map-no_incr'"
timestat MAPPED
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_map.rpt
foreach cg [find /-cost_group*] {
  report timing-cost_group[list $cg] > $_REPORTS_PATH/${DESIGN}_basename$cg_post_map.rpt
}
04/25/11 }
```

Encounter RTL Compiler

57

# The Rest of run.tcl

```
##Intermediate netlist for LEC verification..
write_hdl-lec > ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v
write_do_lec-revised_design${_OUTPUTS_PATH}/${DESIGN}_intermediate.v-logfile ${_LOG_PATH}/rtl2intermediate.lec.log &
${_OUTPUTS_PATH}/rtl2intermediate.lec.do
## ungroup -threshold <value>
#####
## Incremental Synthesis
#####
## Uncomment to remove assigns & insertiehilo cells during Incremental synthesis
##set_attrbuteremove_assignstrue /
##set_remove_assign_options=buffer_or_inverter<libcell> -design <design|subdesign>
##set_attributeuse_tiehilo_for_const<none|duplicate|unique> /
synthesize-to_mapped-eff $MAP_EFF-incr
puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL
foreach cg [find /-cost_group-null_ok*] {
    report timing-cost_group[list $cg] > ${_REPORTS_PATH}/${DESIGN}_$asname$cg]_post_incr.rpt
}
#####
## Spatial mode optimization
#####
## Uncomment to enable spatial mode optimization
##synthesize-to_mapped-spatial
#####
## write Encounter file set (verilog, SDC, config, etc.)
#####
##write_encounterdesign-basename<path & base filename>-lef <lef_file(s)>
report area > ${_REPORTS_PATH}/${DESIGN}_area.rpt
report datapath > ${_REPORTS_PATH}/${DESIGN}_datapath_incr.rpt
report messages > ${_REPORTS_PATH}/${DESIGN}_messages.rpt
report gates > ${_REPORTS_PATH}/${DESIGN}_gates.rpt
write_design-basename${_OUTPUTS_PATH}/${DESIGN}_m
## write_hdl > ${_OUTPUTS_PATH}/${DESIGN}_m.v
## write_script> ${_OUTPUTS_PATH}/${DESIGN}_m.script
write_sdc> ${_OUTPUTS_PATH}/${DESIGN}_m.sdc
#####
### write_do_lec
#####
write_do_lec-golden_design${_OUTPUTS_PATH}/${DESIGN}_intermediate.v-revised_design${_OUTPUTS_PATH}/${DESIGN}_m.v\
-logfile ${_LOG_PATH}/intermediate2final.lec.log > ${_OUTPUTS_PATH}/intermediate2final.lec.do
##Uncomment if the RTL is to be compared with the final netlist..
##write_do_lec-revised_design${_OUTPUTS_PATH}/${DESIGN}_m.v-logfile ${_LOG_PATH}/rtl2final.lec.log > ${_OUTPUTS_PATH}/rtl2final.lec.do
puts "Final Runtime & Memory."
timestat FINAL
```

04/25/11

Encounter RTL Compiler

58

# 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
```

# 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.

You can set **search paths** or aliases or source any Tcl procedures you want loaded from within this file.

## Sample `.synth_init` file

```
source ~/my_file.tcl
set_attr script_search_path "../tcl ." /
set_attr hdl_search_path "../rtl" /
set_attr lib_search_path "../library" /
```

04/25/11

Encounter RTL Compiler

60

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.



# 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_list " 01_wc3.lib mylib1.lib x1.lib " # Variable`  
`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}} /`

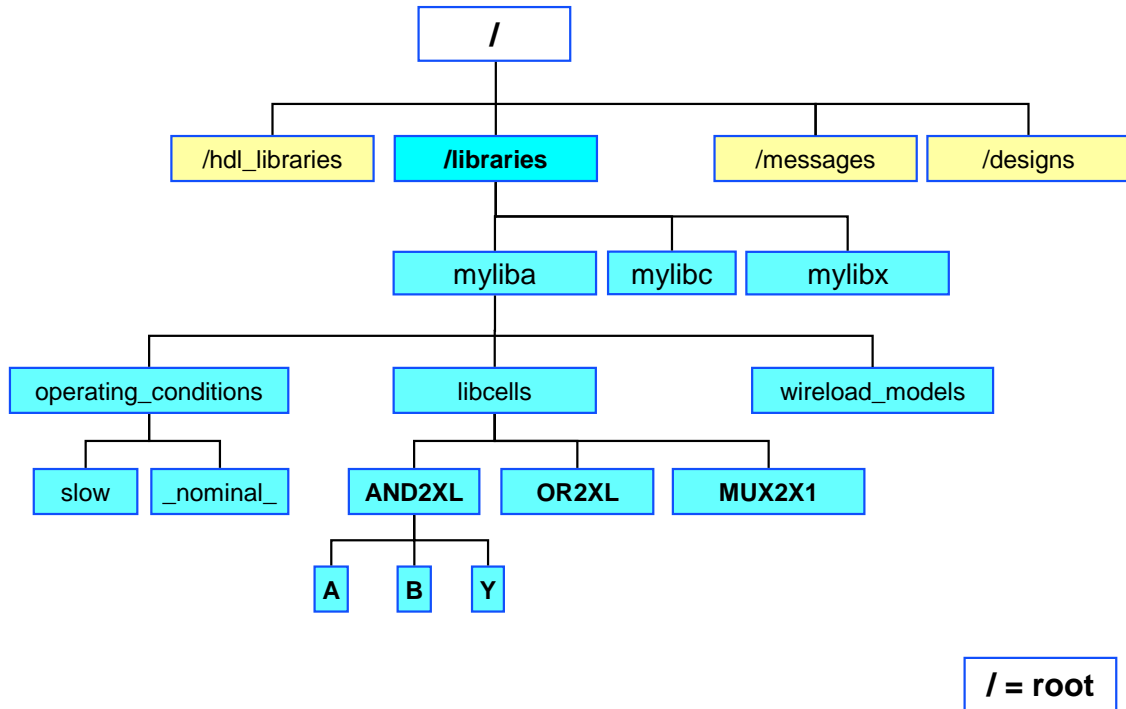
In this example, RC loads *a.lib* and appends *b.lib* to *a.lib*. Next, it loads *c.lib*. Then, it loads *x.lib* and appends *y.lib* to *x.lib*.

- ◆ To set the library search path, enter:  
`set_attribute lib_search_path <path> /`

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

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

# Library Information Hierarchy



04/25/11

Encounter RTL Compiler

62

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/sn1_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.

## 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.

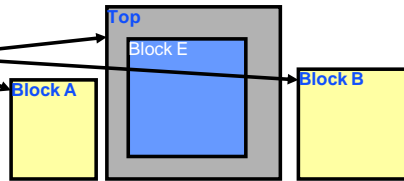
```
set_attr lef_library <lef header>
set_attr cap_table_file <cap table>
// set_attr interconnect_mode ple /
```

Fanout	Load
1	1
2	1
3	2
4	2
5	3
6	4
7	5
8	6
9	7
10	8
10	9
10	10

## 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
```



04/25/11

Encounter RTL Compiler

64

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}
```

-sv Specifies that the HDL files conform to SystemVerilog 3.1.a.

If you do not specify either the -v1995, -v2001, -sv or the -vhdl option, the default language format is that specified by the *hdl\_language* attribute. The default value for the *hdl\_language* attribute is -v1995.

-vhdl Specifies that the HDL files are VHDL files. The *hdl\_vhdl\_read\_version* root attribute value specifies the standard to which the VHDL files conform.

## Reading Designs (continued)

---

To set the HDL search path, enter the following:

```
set_attribute hdl_search_path "path" /
```

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

```
set_attr hdl_language vhd1 /  
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.

Alternatively, to read the gate-level (structural) netlist, use:

```
read_netlist design_struc1.v
```

This command not only parses the netlist but also elaborates the design.

# 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

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

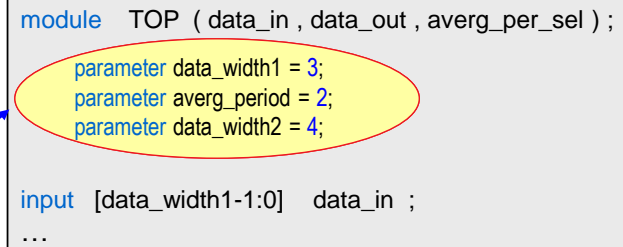
```
elaborate [-h] [-parameters {} ] [<top_module_name>]
```

## 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 ;  
    ...  
endmodule
```

04/25/11

Encounter RTL Compiler

67

- h: Help with the elaborate command
- parameters: Integer list of design parameters (Or use name association)
- [-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.

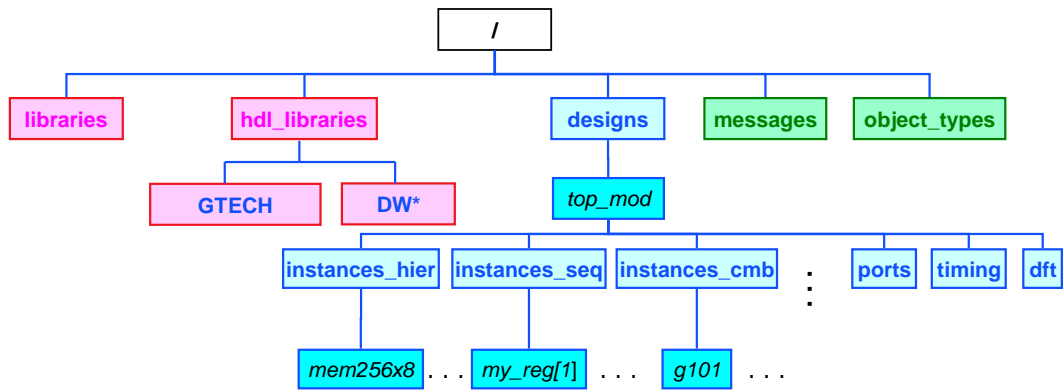
After elaboration finishes, the software reports

§Unresolved references, that is, instances found with no corresponding module or library cell

§Semantic problems, including:

- Read before write
- Unused ports, inconsistent resets

# Design Information Hierarchy



Directory structure maps to the design hierarchy.

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

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

Before elaborating a design, set the following attribute to *true* to keep track of the RTL source code for schematic cross-probing and reporting purposes:

```
set_attribute hdl_track_filename_row_col {true | false}
```



# Checking for Design Issues

Use *check\_design* to check for design problems such as undriven or multidriven 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.

Check Design Report Summary	
-----	
Name	Total
-----	
Unresolved References	2
Empty Modules	0
Unloaded Port(s)	15
Unloaded Sequential Pin(s)	619
Assigns	36503
Undriven Port(s)	0
Undriven Leaf Pin(s)	0
Undriven hierarchical pin(s)	0
Multidriven Port(s)	3
Multidriven Leaf Pin(s)	0
Multidriven hierarchical Pin(s)	11
Constant Port(s)	0
Constant Leaf Pin(s)	11184
Constant hierarchical Pin(s)	8885

# 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.

You can look for failed commands using:

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

- ❑ 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.

04/25/11

Encounter RTL Compiler

70

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).

# Problems with Reading SDC Files

Check the log file for errors and warnings.

Edit all unsupported commands and reenter them using the interactive `dc::` command.

You **cannot** cut-and-paste a Synopsys Design Constraint file line-by-line *unless* you precede each SDC command with "`dc::`"

```
rc:/> dc::create_clock -period  
1.0 [dc::get_ports clk]
```

```
Warning : Unsupported SDC command option. [SDC201] [set_units]
: The 'set_units' command on line '8' in the SDC file 'top.sdc' does not support the '
resistance', 'voltage', 'current' and the 'power' options.
: The current version does not support this SDC command option. However, future versions
may be enhanced to support this option.
Error : Could not interpret SDC command. [SDC202] [read_sdc]
: The 'read_sdc' command encountered an error while processing this command on line '8'
of the SDC file 'top.sdc':set_units-time ns-resistancekOhm-capacitance pF-voltage V-current
mA.
: The 'read_sdc' command encountered a problem while trying to evaluate an SDC
command. This SDC command will be added to the Tcl variable $::d$dc_failed_commands
```

Statistics for commands executed by `read_sdc`:

"create_clock"	- successful	7 , failed	0 (runtime 2.00)
"current_design"	- successful	2 , failed	0 (runtime 0.00)
"get_cells"	- successful	1626 , failed	0 (runtime 1.00)
"get_clocks"	- successful	127 , failed	0 (runtime 0.00)
"get_pins"	- successful	54971 , failed	0 (runtime 30.00)
"get_ports"	- successful	77536 , failed	0 (runtime 25.00)
"group_path"	- successful	6 , failed	0 (runtime 1.00)
"set_case_analysis"	- successful	3 , failed	0 (runtime 0.00)
"set_clock_groups"	- successful	1 , failed	0 (runtime 1.00)
"set_clock_uncertainty"	- successful	14 , failed	0 (runtime 0.00)
"set_false_path"	- successful	1113 , failed	0 (runtime 11.00)
"set_ideal_network"	- successful	36 , failed	0 (runtime 0.00)
"set_input_delay"	- successful	8066 , failed	0 (runtime 11.00)
"set_input_transition"	- successful	15598 , failed	0 (runtime 9.00)
"set_load"	- successful	8962 , failed	0 (runtime 2.00)
"set_max_area"	- successful	1 , failed	0 (runtime 0.00)
"set_max_delay"	- successful	6 , failed	0 (runtime 0.00)
"set_max_fanout"	- successful	1 , failed	0 (runtime 0.00)
"set_max_transition"	- successful	1 , failed	0 (runtime 0.00)
"set_multicycle_path"	- successful	173 , failed	0 (runtime 0.00)
"set_operating_conditions"	- successful	0 , failed	1 (runtime 0.00)
"set_output_delay"	- successful	8962 , failed	0 (runtime 9.00)
"set_propagated_clock"	- successful	7 , failed	0 (runtime 0.00)
"set_units"	- successful	0 , failed	1 (runtime 0.00)

**Warning : Total failed commands during read\_sdc are 2**

# 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
Unlocked primary I/Os	Define input/output delay for these I/Os.
Unlocked flops	Check the fanin cone of these flops using the <i>fanin</i> command.
Multiple clocks propagating to the same sequential clock pin	To see which clocks are being propagated to that pin, use the <i>inverting_clocks</i> or <i>non_inverting_clocks</i> attribute of the pin. Use the <i>timing_case_logic_value</i> attribute to propagate only one clock to that pin ( <i>set_case_analysis</i> ).
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.

04/25/11

Encounter RTL Compiler

72

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

# Problems in logfile: report timing -lint

---

-----  
The following sequential clock pins are either unconnected or driven by a logic constant:

/designs/top/instances\_hier/Chip/instances\_seq/U\_SPARE\_FF\_0/pins\_in/clock  
... 15 other warnings in this category.

-----  
The following sequential data pins are driven by a clock signal:

/designs/top/instances\_hier/Core/instances\_seq/CU\_wrapper\_0/pins\_in/CU\_CLK  
... 7 other warnings in this category.

-----  
The following sequential clock pins have no clock waveform driving them. No timing constraints will be derived for paths leading to or from these pins.

/designs/top/instances\_hier/Core/instances\_seq/q\_reg/pins\_in/clock  
... 35 other warnings in this category.

-----  
The following sequential clock pins have multiple clock waveforms from a single timing mode driving them. Timing analysis will use all of the waveforms to determine the most pessimistic timing constraints. The pin attribute 'propagated\_clocks' can be used to analyze which clocks are being used at each clock pin.

/designs/top/instances\_hier/Core/instances\_seq/q\_reg[0]/pins\_in/clock  
... 8188 other warnings in this category.

# 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>* — carrysave 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 Encounter RTL Compiler.

# Logfile Exclusive: Global Optimization Targets

A target is printed for every clock domain & cost group specified.

Always check the target #s before proceeding further. Positive Target is Good.

If target is too negative then either something is wrong in the constraints or there is bug in the tool. Fix the constraint.

Target drives the tool to optimize those paths.

Path adjust your I/Os so they are not showing up in target; false paths & multicycle paths shouldn't show up either. (We'll discuss path\_adjust later)

## Global mapping target info

=====  
Cost Group 'qin\_tck' target slack: 292 ps  
Target path end-point (Port: toptdo\_en)

Warning : Possible timing problems have been detected in this design. [TIM-11]  
: The design is 'top'.

Pin	Type	Fanout	Load (fF)	Arrival (ps)
(clock qin_tck)	<<< launch			0 R
u_tap_U8/CP				
u_tap_U8/Q	(u) (P) SDFFP	7	35.0	
cb_oseq/U1_Q				
g24748/in_1				
g24748/z				
tdo_en	<<< unmapped_or2	1	101.8	
(top.sdc_line_18407)	out port ext delay			
(clock qin_tck)	capture			25000
F	uncertainty			

=====  
Cost Group : 'qin\_tck' (path\_group'qin\_tck')  
Start-point : u\_tap\_U8/CP  
End-point : tdo\_en

The global mapper estimates a slack for this path of 957ps.

# Reporting

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



# Generating Timing Reports

---

## Syntax

```
report timing [-endpoints] [-summary] [-lint] [-verbose]
  [-full_pin_names] [-physical] [-gtd] [-gui] [-num_paths <integer>]
  [-worst <integer>] [-slack_limit <delay in picoseconds>]
  [-mode <mode>]
  [-from <instance|external_delay|clock|port|pin>+]
  [-through <instance|port|pin>+]+
  [-to <instance|external_delay|clock|port|pin>+] [-paths <string>]
  [-exceptions <exception>+] [-cost_group <cost_group>+] [> file]
```

### **puts "WNS path/timing among all in2out paths"**

```
report timing -from [all des inps] -to [all des outs]
```

### **puts "WNS path/timing among all reg2reg paths"**

```
report timing -from [all des seqs] -to [all des seqs]
```

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

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

The "all" is from the Tcl procedures under /<path\_to\_installation>/etc/ .

# Reading Timing Reports

```

=====
Generated by:      Encounter(R) RTL Compiler v09.10-s203_1
Generated on:      Aug 20 2010 03:30:25 PM
Module:           dtmf_recvr_core
Technology libraries: ss_g_lv08_125c 1.1, ram 256x16_slow 1.1,
                   rom 512x16A_slow 1.1, pllclk 4.3
Operating conditions: ss_lv08_125c
Interconnect mode:  global
Area mode:        timing library
=====

```

Header includes library and module information.

Pin	Type	Fanout	Load	Slew	Delay	Arrival	
					(fF)	(ps)	(ps)
(clock m_clk)	launch						0 R
TDSP_CORE_INST							
DECODE_INST							
ir_reg[8]/clk	(u) unmapped_d_flop		79	56.7	0	+415	0 R
DECODE_INST/ir[8]							415 R
TDSP_CORE_GLUE_INST/ir[8]							
sll_120_49/SH[0]						+0	415
gl/in_0							
...							
inc_add_63_54_6/Z[31]						+0	6031
mux_63_28_g1/data1							
mux_63_28_g1/z	(u) unmapped_bmux3		1	6.3	0	+160	6191 R
MPY_32_INST/result[31]							
EXECUTE_INST/mpy_result[31]							
p_reg[31]/d	unmapped_d_flop					+0	6191
p_reg[31]/clk	setup				0	+135	6326 R
(clock m_clk)	capture						6000 R

Body includes arrival time calculation.

```

-----
Cost Group   : 'm_clk' (path_group 'm_clk')
Timing slack : -326ps (TIMING VIOLATION)
Start-point  : TDSP_CORE_INST/DECODE_INST/ir_reg[8]/clk
End-point    : TDSP_CORE_INST/EXECUTE_INST/p_reg[31]/d
-----

```

Footer includes timing slack calculation.

# 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
```

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.

# 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.

# Filtering Objects by Name and Type

Purpose	Command
Finding all flops and latches	<code>find /des* -instance inst*seq*/*</code>
Finding all the input ports	<code>find /des* -port ports_in/*</code>
Finding all the output ports	<code>find /des* -port ports_out/*</code>
Finding all pins in a hierarchical instance v1	<code>find /des* -pin inst*hier/v1/*</code> <code>find /des* -pin v1/*</code>
Finding all subdesigns.	<code>find /des* -subd *</code>
Finding all hierarchical instances in the design	<code>set inst_list [find /des* -instance inst*hier/*]</code> <code>echo \$inst_list</code>
Finding all clocks in the design	<code>find /des* -clock *</code>
Finding all available wire-load models in a library	<code>find /LibraryName -wireload *</code>

04/25/11

Encounter RTL Compiler

81

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	<code>filter latch true [find / -instance *]</code>
Finding all the preserved instances	<code>filter preserve true [find / -inst *]</code>
Listing all messages with a count of one or more	<code>ls -la [filter -invert count 0 [find / -message *]]</code>

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

```
filter [-invert][-special] [-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 4-1 Running the Basic Synthesis Flow

- Starting the Compiler
- Setting Up the Environment
- Loading Libraries and Designs
- Resolving Unresolved References

## Lab 4-2 Navigating the Design Hierarchy

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

## Lab 4-3 Reading SDC Constraints

- Debugging Failed SDC Constraints
- Analyzing Missing SDC Constraints

## Lab 4-4 Using the Graphical Interface

- Synthesizing Your Design
- Viewing Logical Hierarchy, HDL, and Schematic
- Generating Reports



# Datapath Synthesis

## Module 5

April 25, 2011



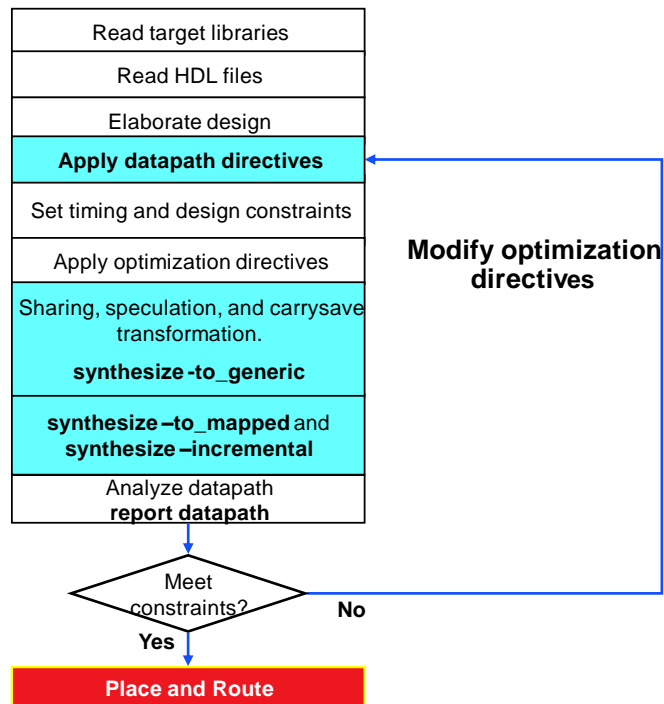
# Module Objectives

---

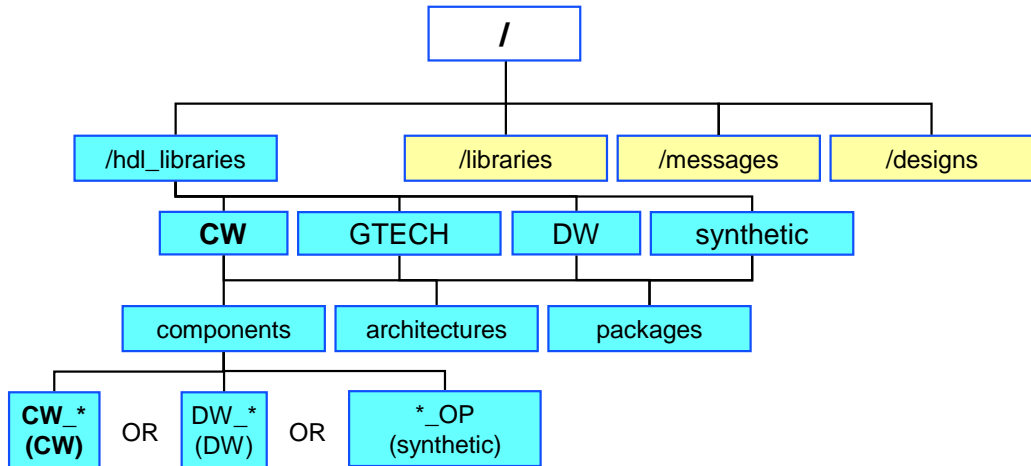
In this module, you

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

# Datapath Flow



# Datapath Information Hierarchy



**/ = root**

04/25/11

Encounter RTL Compiler

88

The following functions are part of the datapath package:

- ChipWare Components, DesignWare, 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)

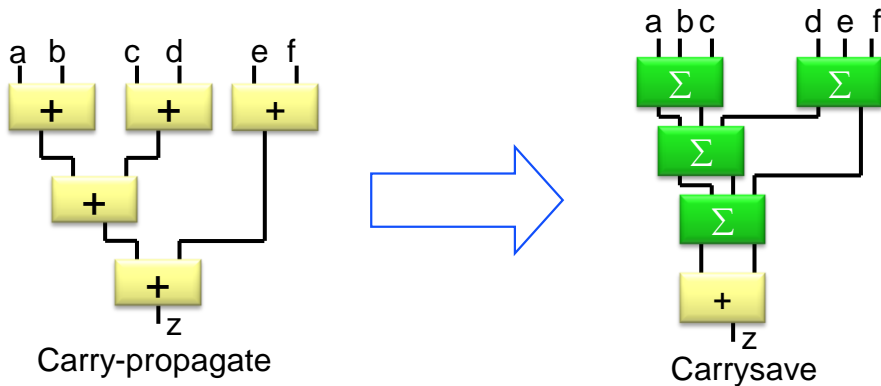
# Generic Optimizations

---

- ◆ An optimization technique is considered generic when the technique has no connection with the library components required for the implementation of the design.
- ◆ Technology-independent mapping and optimization techniques:
  - ❑ Carry save arithmetic optimization
  - ❑ Logic pruning
  - ❑ Resource sharing
  - ❑ Implementation selection
  - ❑ Redundancy removal
  - ❑ Multiplexer optimization
  - ❑ Arithmetic optimization
  - ❑ Common subexpression sharing

# Carry-Save Arithmetic Operations

- ◆ Carry-save arithmetic (CSA) operations are functionally equivalent to their carry-propagate counterparts.
- ◆ The carry logic for the intermediate sums is saved until the very end, thus reducing area and improving timing.



04/25/11

Encounter RTL Compiler

91

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

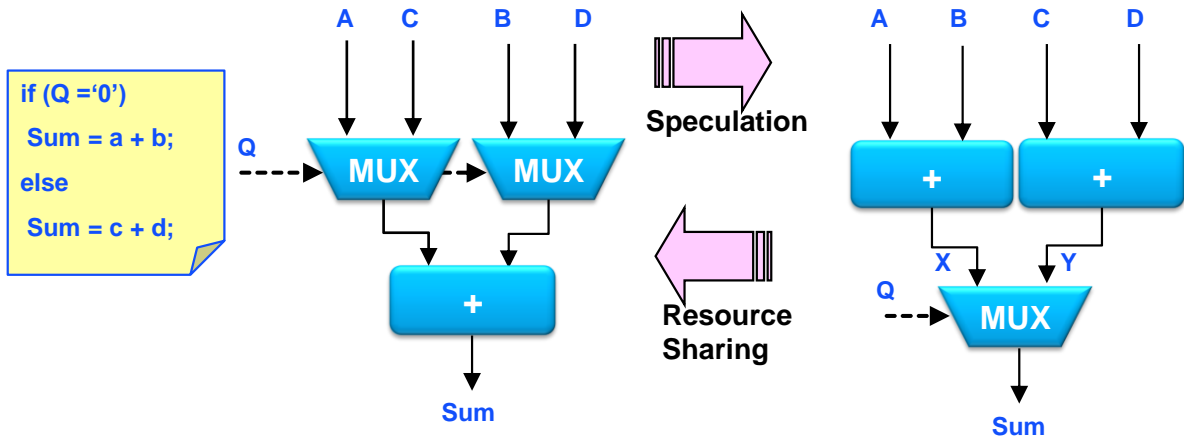
```
wire [16:0] tmp = a + b; assign z = ~tmp + c;
```

- Truncated CSA

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

# Resource Sharing and Speculation

The sharing and un-sharing (speculation) of resources trades off area versus timing during logic synthesis.



04/25/11

Encounter RTL Compiler

92

- § A resource is any computational element, such as an add, shift, or multiplier operation.
- § Each type of operator in the RTL description requires a unique resource type.

For instance,

+ operator requires an adder;

> requires a comparator

- § Maximum number of resources required for each operator type is the number of times an operator is used in the RTL description.
- § However, resources can be reduced through sharing, thus saving area.

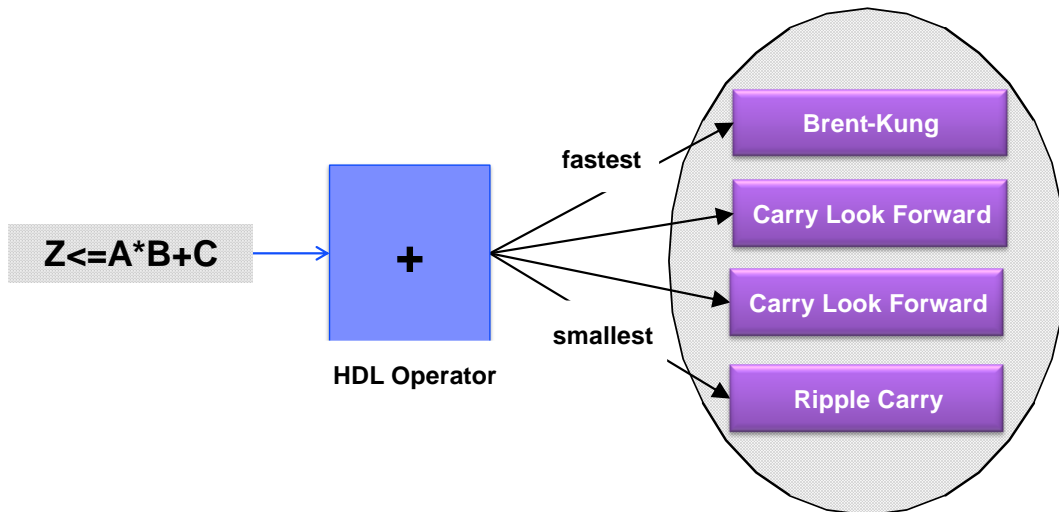
Some operators can be mapped to a common resource type. For instance, + and - operators can be mapped to an add-subtract unit.

- § Operators in different clock cycles can share the same resource. This is determined by analyzing if there are any data flow or control flow conflicts.



# Implementation Selection: Architecture Tradeoff

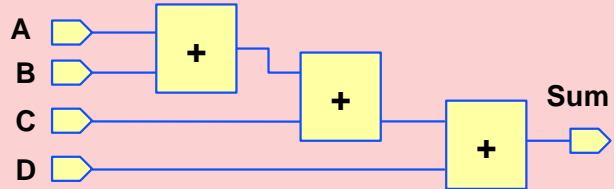
- ◆ Different implementations of the design components have different area and timing characteristics.
- ◆ Design constraints determine the appropriate design component.



# Arithmetic Optimization

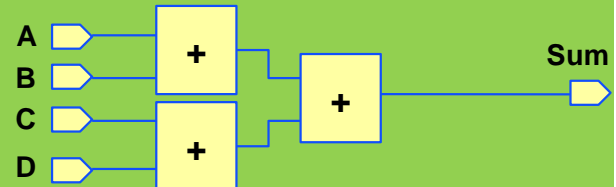
$$\text{Sum} \leq A + B + C + D$$

**Initial Order**



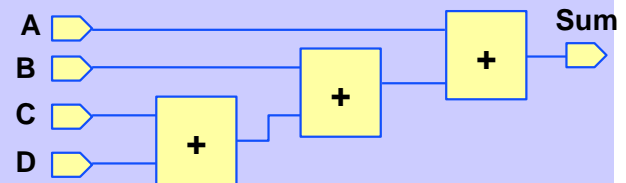
**Optimized for Speed**

- All inputs have equal delay.



**Optimized for Speed**

- Input A is late arriving.



**Note:** Operators cannot be rearranged if the initial order is overridden by parentheses in HDL.

# Common Subexpression Sharing

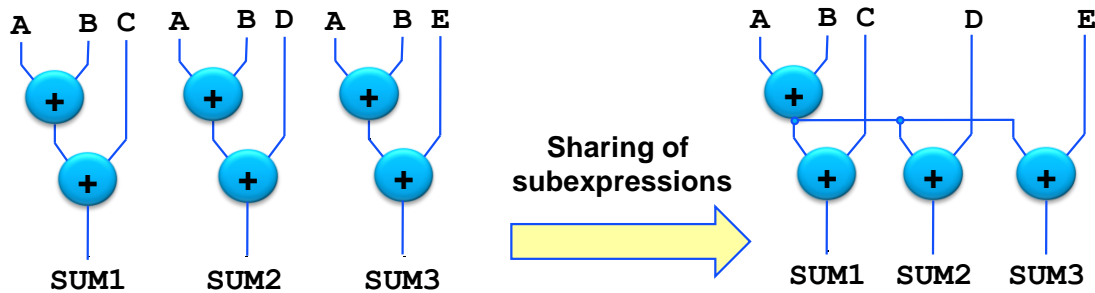
Consider the assignments:

`SUM1 <= A + B + C`

`SUM2 <= A + B + D`

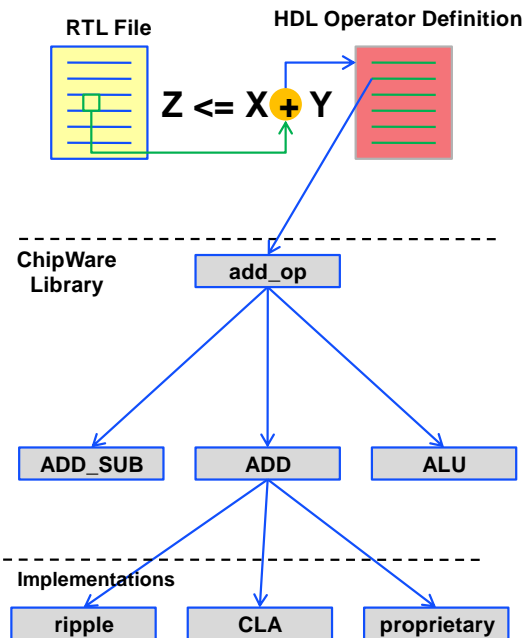
`SUM3 <= B + A + E`

The “A+B” subexpression can be shared, thus saving two adders in the process. The order within the subexpression is not important, but the position must be the same.



# Implementation Selection: ChipWare

- ◆ Some advanced synthesis tools come with a library of reusable designs.
  - ❑ Cadence Encounter RTL Compiler (RC) has such a library, known as ChipWare.
- ◆ ChipWare (CW) library includes
  - ❑ Common combinational and sequential components
  - ❑ Arithmetic components (adders, subtractors, multipliers)
  - ❑ Memory components (flip flops, FIFOs)
- ◆ Logic synthesis searches for operators in RTL files it reads and automatically maps those operators to CW components, if available.
- ◆ CW components often have multiple architectural implementations that allow logic synthesis to pick one according to design need.



# Commands for Technology-Independent Mapping

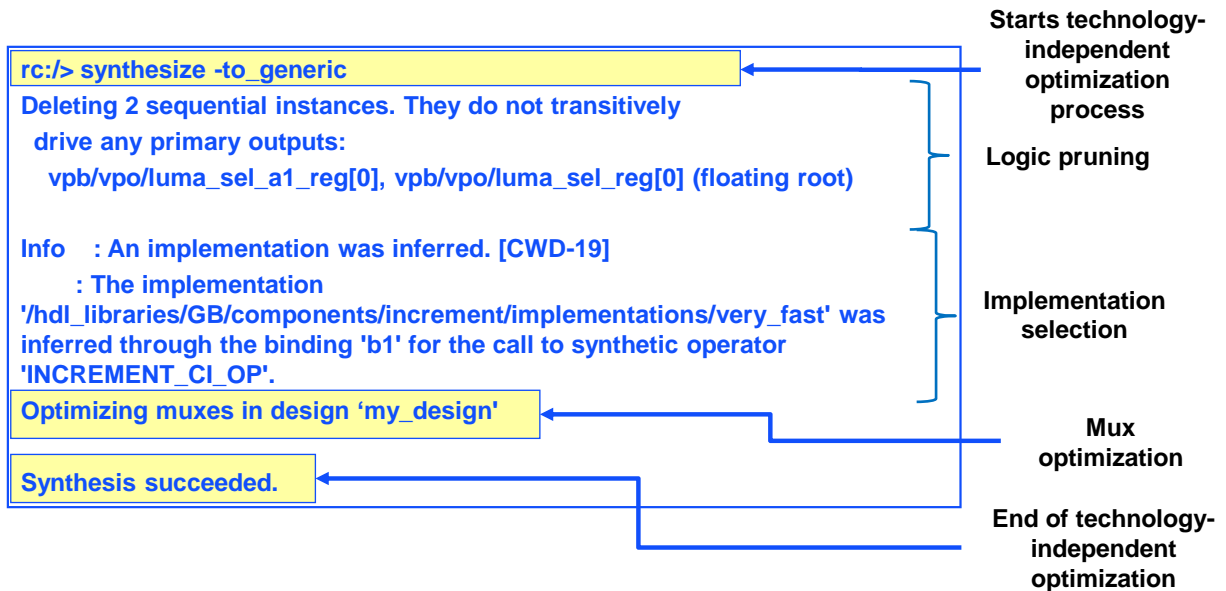
---

- ◆ In this stage, logic synthesis performs technology-independent optimizations, including
  - ❑ Constant propagation
  - ❑ Resource sharing
  - ❑ Logic speculation
  - ❑ Multiplexor optimization
  - ❑ Carry-save arithmetic optimization
- ◆ You can run this stage separately by using the following command:  
`synthesize -to_generic -effort <effort_level>`

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

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

# Log Entries for Technology Independent Mapping



04/25/11

Encounter RTL Compiler

98

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.

# 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:

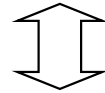
```
set_attr speed_grade speed  
[find /designs* -subdesign name]
```

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 speed_grade  
[find / -subdesign <name>]
```

Faster/  
Larger



Smaller/  
Slower



**Speed Grading**

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

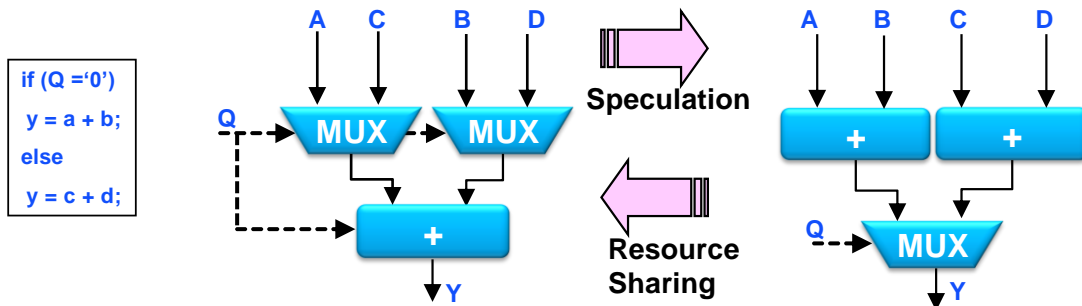
# Sharing and Speculation

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_sharing advanced|basic|none /
```

```
set_attr dp_speculation basic|none /
```



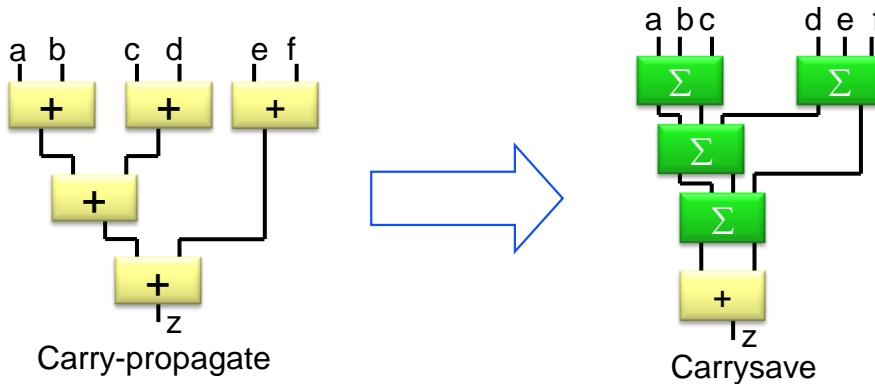


# Carrysave Arithmetic Operations

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_csa none / ## Default is basic
```



04/25/11

Encounter RTL Compiler

101

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;
```

- 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 speculation on subdesigns, enter:

```
set_attr dp_speculation {inherited|basic|none}  
[find / -subdesign name]
```

To control CSA on subdesigns, enter:

```
set_attr dp_csa {inherited|basic|none}  
[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 *carriesave 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.

# Datapath Rewriting

---

Datapath rewriting performs QOR-driven transformations from one group of datapath operators to another, as if the RTL code were rewritten. This is turned on by default during *synthesize -to\_generic -effort high*.

If you want to control datapath rewriting manually, use the commands shown here.

- ◆ To turn on, use:
  - `set_attr dp_rewriting basic /`
- ◆ To turn off, use:
  - `set_attr dp_rewriting none /`

## LEC Impact

- ◆ No problem

# Example: Datapath Rewriting

---

## Sample RTL Code

```
module tst (y, a, b, s);  
    input s;  
    input [15:0] a, b;  
    wire [15:0] p, q;  
    output [15:0] y;  
    assign p = a - b;  
    assign q = a + b;  
    assign y = s ? p : q;  
endmodule
```

## Datapath Rewritten

```
module tst (y, a, b, s);  
    input s;  
    input [15:0] a, b;  
    wire [15:0] t;  
    output [15:0] y;  
    assign t = {16{s}} ^ b;  
    assign y = a + t + s;  
endmodule
```

## QOR Improvement\*

```
OPT OFF: period = 7500, slack = 12.0, area = 482.0  
OPT ON : period = 7500, slack = 52.9, area = 267.0  
==> 44.6% smaller
```

# 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						

04/25/11

Encounter RTL Compiler

106

Use the *report datapath* command to:

- Identify datapath operators.
- Examine how carriesave arithmetic 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*.

# Lab Exercises

---



## Lab 5-1 Running Datapath Synthesis

---

Blank Page



# Optimization Strategies

## Module 6



April 25, 2011

# 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

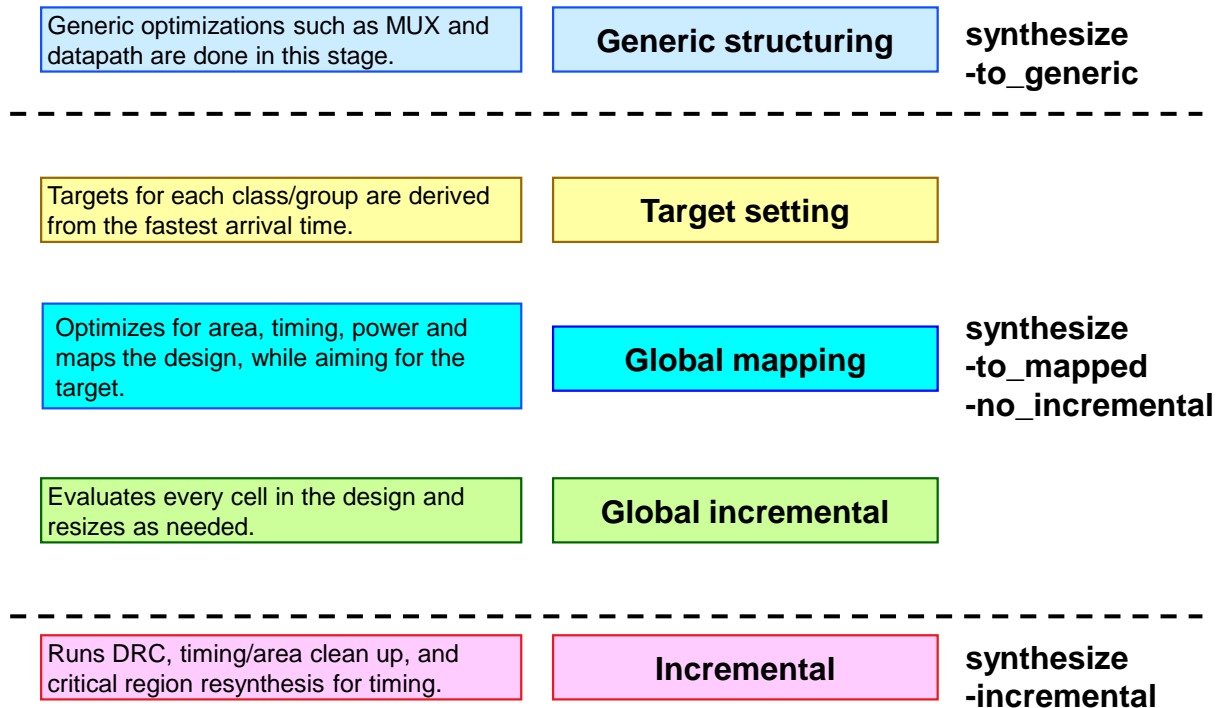
# Technology Transformation (Mapping)

---

- ◆ Technology transformation or “technology mapping” is the phase of logic synthesis when gates are selected from a technology library to implement the circuit.
- ◆ Why run technology mapping and related optimizations?
  - ❑ Gates in the library are pre-designed and are usually optimized for various combinations of area, delay, and power.
    - Use fastest gates along the critical path, and
    - Use area-efficient gates (or a combination) off the critical path
- ◆ Technology mapping is normally done after technology-independent optimization (unless you have a gate-level netlist, in which case you would skip technology-independent optimization).

This module primarily deals with technology mapping related optimizations.

# Synthesis Stages



04/25/11

Encounter RTL Compiler

112

## § Target setting

Target timing goals (clock period) for each class and group of timing paths are derived from the fastest arrival time.

## § Global mapping

Optimizes for area, timing, power, and maps the design while aiming for the target clock frequency

## § Global incremental

Evaluates every cell in the design and resizes as needed to improve area and power consumption

## § Incremental optimization

Runs design rule checks (DRCs), timing and area cleanup, and critical region resynthesis (CRR) for timing optimization

# 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.

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 loading a Verilog® mapped netlist, do not use this command unless it is absolutely necessary, because it will unmap 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...
```

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

# Synthesis Stages: Global Mapping Target Report

## Global mapping target info

=====  
Cost Group 'm\_clk' target slack: 224 ps

Pin	Type	Fanout	Load	Arrival (ff)	(ps)
-----					
(clock m_clk)	<<< launch				0 R
DECODE_INST					
cb_seqi					
ir_reg[8]/clk					
ir_reg[8]/q	(u)	unmapped_d_flop	90	63.0	
cb_seqi/ir[8]					
DECODE_INST/ir[8]					
...					
EXECUTE_INST/mpy_result[31]					
cb_seqi/mpy_result[31]					
g22038/data1					
g22038/z	(u)	unmapped_bmux3	1	6.3	
p_reg[31]/d	<<<	unmapped_d_flop			
p_reg[31]/clk		setup			
-----					
(clock m_clk)	capture				8000 R
-----					

Cost Group : 'm\_clk' (path\_group 'm\_clk')  
Start-point : TDSP\_CORE\_INST/DECODE\_INST/cb\_seqi/ir\_reg[8]/clk  
End-point : TDSP\_CORE\_INST/EXECUTE\_INST/cb\_seqi/p\_reg[31]/d  
(u) : Net has unmapped pin(s).

The global mapper estimates a slack for this path of 589ps.

04/25/11

Encounter RTL Compiler

115

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**

```
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...
```

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

## Global mapping timing result

Pin	Type	Fanout	Load	Slew	Delay	Arrival	
			(fF)	(ps)	(ps)	(ps)	(ps)
(clock m_clk)	launch						0 R
TDSP_CORE_INST							
DECODE_INST							
cb_seqi							
ir_reg[8]/CK					0		0 R
ir_reg[8]/Q	SDDFFSHQX4M		5	64.8	337	+384	384 R
cb_seqi/ir[8]							
DECODE_INST/ir[8]							
EXECUTE_INST/alu_result[14]							
cb_seqi/alu_result[14]							
acc_reg[14]/SI <<<	SDDFFQX2MTH					+0	6381
acc_reg[14]/CK	setup				0	+897	7278 R
(clock m_clk)	capture						8000 R
Cost Group : 'm_clk' (path_group 'm_clk')							

## Timing slack : 722ps

Start-point : TDSP\_CORE\_INST/DECODE\_INST/cb\_seqi/ir\_reg[8]/CK  
 End-point : TDSP\_CORE\_INST/EXECUTE\_INST/cb\_seqi/acc\_reg[14]/SI

04/25/11

Encounter RTL Compiler

117

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: Global Incremental

In this stage RC refines the timing and area of all critical paths by remapping the cells according to the new surroundings of the cell.

## Global incremental optimization status

Operation	Total Area	Group		
		Total	Worst	
		Slacks	Worst	Path
-----				
global_inc	252520	-89		
				TDSP_CORE_INST/EXECUTE_INST/sel_op_a_reg[1]/CK-->
				TDSP_CORE_INST/EXECUTE_INST/p_reg[31]/D

# Stages: Global Incremental Target Report

## Global incremental target info

Cost Group 'm\_clk' target slack: 142 ps

Pin	Type	Fanout (fF)	Load	Arrival (ps)
(clock m_clk)	<<< launch			0 R
TDSP_CORE_INST				
DECODE_INST				
cb_seqi				
ir_reg[8]/CK				
ir_reg[8]/Q	SDDFFSHQX4M	5	64.8	
cb_seqi/ir[8]				
DECODE_INST/ir[8]				
EXECUTE_INST/alu_result[14]				
cb_seqi/alu_result[14]				
acc_reg[14]/SI <<< SDDFFQX2MTH				
acc_reg[14]/CK	setup			
(clock m_clk)	capture			8000 R

Cost Group : 'm\_clk' (path group 'm\_clk')

Start-point : TDSP\_CORE\_INST/DECODE\_INST/cb\_seqi/ir\_reg[8]/CK

End-point : TDSP\_CORE\_INST/EXECUTE\_INST/cb\_seqi/acc\_reg[14]/SI

The global mapper estimates a slack for this path of 626ps.

# Synthesis Stages: Incremental Synthesis

Run *synthesize -incremental* to fix any timing issues and clean up DRC violations.

```
Incremental optimization status
=====
```

Operation	Total Area	Group				DRC Totals	
		Total Worst Slacks	Max Trans	Max Cap	Max Fanout		
init_iopt	252520	-89	0	0	16		
Path: TDSP_CORE_INST/EXECUTE_INST/sel_op_a_reg[1]/CK --> TDSP_CORE_INST/EXECUTE_INST/p_reg[31]/D							
simp_cc_in	252271	-78	0	0	16		
Path: TDSP_CORE_INST/EXECUTE_INST/sel_op_b_reg[0]/CK --> TDSP_CORE_INST/EXECUTE_INST/p_reg[31]/D							
-----							
Operation	Total Area	Group				DRC Totals	
		Total Worst Slacks	Max Trans	Max Cap	Max Fanout		
init_delay	252271	-78	0	0	16		
Path: TDSP_CORE_INST/EXECUTE_INST/sel_op_b_reg[0]/CK --> TDSP_CORE_INST/EXECUTE_INST/p_reg[31]/D							
incr_delay	252393	0	0	0	16		
init_drc	252393	0	0	0	16		
incr_drc	252432	0	0	0	3		
incr_drc	252443	0	0	0	0		

04/25/11

Encounter RTL Compiler

120

## DRC: Design Rule Checks

The *drc\_first* attribute specifies whether to give all the design rule constraints higher priority than the timing constraints.

## Incremental Synthesis:

- Works with an already mapped design.
- Incrementally optimizes the mapped design.

Allows the mapper to preserve the current implementation of the design and perform incremental optimizations if and only if the procedure guarantees an improvement in the overall cost of the design.

- Can be run as many times as needed.

# Synthesis Stages: Incremental Synthesis (continued)

Run *synthesize -to\_mapped –incremental* to achieve power goals.

Operation	Group					Leakage Power	Switching Power
	Total Area	Worst Slacks	Max Trans	Max Cap	Max Fanout		
init_power	252443	0	0	0	0	42674	14210023
p_rem_buf	252160	0	0	0	0	42309	14190812
p_rem_inv	252078	0	0	0	0	42277	14186008
p_merge_bi	252002	0	0	0	0	42207	14182209
io_phase	251987	0	0	0	0	42196	14179410
gate_comp	251338	0	0	0	0	41717	14104759
glob_power	250828	0	0	0	0	39950	14027472
power_down	250189	0	0	0	0	36940	13937300
p_rem_buf	250181	0	0	0	0	36911	13936989
p_merge_bi	250170	0	0	0	0	36910	13936446

CRR: Critical region resynthesis

# Setting Effort Levels

---

You can specify an effort level by using the *-effort {low | medium | high}* option with the *synthesize* command. The possible values for the *-effort* option are:

- ◆ **Low:** The design is mapped to gates, but RC does very little RTL optimization, incremental clean up, or redundancy identification and removal. The low setting is generally not recommended.
- ◆ **Medium** (default setting): RC performs better timing-driven structuring, incremental synthesis, and redundancy identification and removal on the design.
- ◆ **High:** RC 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.

# 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.
- ◆ You can validate your timing constraints, with the Encounter® Conformal® Constraint Designer (CCD) tool, before you apply them to your design. Use the *write\_do\_ccd* command to write out a CCD dofile.  
*write\_do\_ccd validate -sdc list\_of\_SDC\_files > Dofile*  
The generated dofile can be then used to run CCD to perform checks on the constraints.



# 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.
- ◆ Set the *dp\_area\_mode* and *dp\_postmap\_downsize* attributes to *true* if you have datapath in your design.
- ◆ 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.

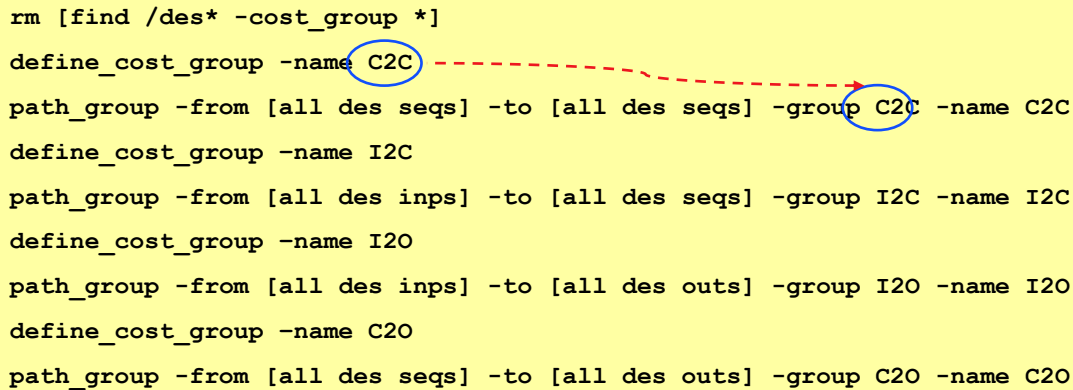
PLE: Physical Layout Estimation

# 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 des seqs] -to [all des seqs] -group C2C -name C2C
define_cost_group -name I2C
path_group -from [all des inps] -to [all des seqs] -group I2C -name I2C
define_cost_group -name I2O
path_group -from [all des inps] -to [all des outs] -group I2O -name I2O
define_cost_group -name C2O
path_group -from [all des seqs] -to [all des outs] -group C2O -name C2O
```



04/25/11

Encounter RTL Compiler

126

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

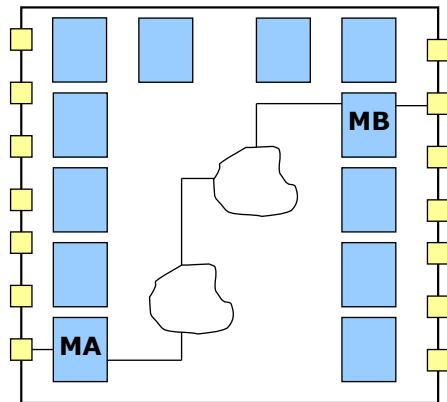
Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)
(clock CKG_VIT_CLK)	<<< launch					0 R
VIT_ACS0						
...						
ACS_DEC0[1]	<<< out port					
(RC_vit_acsu_32.sdc_line_1551)	ext delay					
(clock CKG_VIT_CLK)	capture					800 R
<b>Cost Group : 'sysclk' (path_group 'sysclk')</b>						
Start-point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

# Tightening Constraints

Path adjust command can be used to tighten (or loosen) the timing constraint on a path or set of paths.

**path\_adjust -from <object(s)> -to <object(s)> -delay <delay\_value>**

- ❑ Assume that in RTL code, MA, MB, and logic are all in the same block
- ❑ During synthesis, this appears to be an easy path to satisfy
- ❑ In reality, MA, MB, and logic are spread across the die
- ❑ path\_adjust can be used to make path from MA to MB appear more critical in synthesis
- ❑ Advantage of using path\_adjust is that it is not written out in the SDC constraints, so P&R tool sees the “real” timing



**rc:/> path\_adjust -from MA -to MB -delay -100**

## Tightening Constraints (continued)

---

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 des seqs] -to [all des  
seqs] -name pa_c2c  
path_adjust -delay 200 -from [all des inps] -to [all des  
seqs] -name pa_i2c  
path_adjust -delay 200 -from [all des seqs] -to [all des  
outs] -name pa_c2o  
path_adjust -delay 200 -from [all des inps] -to [all des  
outs] -name pa_i2o
```

Remember to remove *path\_adjusts* before timing reports to normalize timing reports:

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

# Tightening Constraints (continued)

## Path Adjust: STA Using report timing

```
Generated by: Encounter RTL Compiler 10.1.0-S200_1
Generated on: Aug 24 2010 12:27:06 AM
Module: dtmf_recvr_core
Technology libraries: ss_g_1v08_125c 1.1
                   ss_hvt_1v08_125c 1.1
                   ram_256x16_slow 1.1
                   rom_512x16A_slow 1.1
                   pllclk 4.3
Operating conditions: ss_1v08_125c
Interconnect mode: global
Area mode: timing library
```

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

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)	
(clock m_clk)	launch					0	R
TDSP_CORE_INST							
EXECUTE_INST							
sel_op_a_reg[1]/CK				0		0	R
sel_op_a_reg[1]/Q	DFFQX4MTH	3	27.2	205	+583	583	F
EXECUTE_INST/sel_op_a[1]							
...							
EXECUTE_INST/mpy_result[31]							
p_reg[31]/D	EDFFHQX1MTH			+0		7701	
p_reg[31]/CK	setup			0	+388	8089	R
(clock m_clk)	capture					8000	R
adjustments				-200		7800	

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

R – rising edge

F – falling edge

Clock capture time (period)

```
Exception : 'path_adjusts/myPA' path adjust -200ps
Cost Group : 'm_clk' (path_group 'm_clk')
Timing slack : -189ps (TIMING VIOLATION)
Start-point : TDSP_CORE_INST/EXECUTE_INST/sel_op_a_reg[1]/CK
End-point : TDSP_CORE_INST/EXECUTE_INST/
```

Exception created by path\_adjust

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

04/25/11

Encounter RTL Compiler

130

After the constraints are “normalized” by removing the path adjusts, the timing report will show the actual slack.

In this case, the slack will be positive slack after normalizing the constraints.

# 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

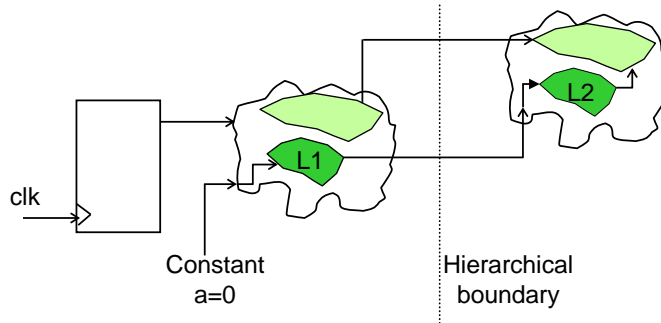
During boundary optimization, RC will push constants into and out of hierarchy, and rewire feedthroughs and complimentary signals. By default, boundary optimization is performed.

To turn off boundary optimization on subdesigns, use:

```
set_attr boundary_opto false [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.

04/25/11

Encounter RTL Compiler

132

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.

To check which type of boundary optimization was performed on a pin use the `boundary_change pin` attribute:

```
get_attribute boundary_change [find /des* -pin name]
```

You can control boundary optimization during synthesis using the following attributes:

- `boundary_opto`
- `boundary_optimize_constant_hier_pins`
- `boundary_optimize_equal_opposite_hier_pins`
- `boundary_optimize_feedthrough_hier_pins`
- `boundary_optimize_invert_hier_pins`



# 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 (Dangling logic can be caused by boundary optimization). 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.

# Sequential Merging

---

Sequential merging combines equivalent sequential cells, both flops and latches, within the same hierarchy.

- ◆ Root attributes

```
set_attr optimize_merge_flops true / ## Default
set_attr optimize_merge_latches true /      ## Default
```

- ◆ Finer grained instance-level control

```
set_attr optimize_merge_seq true [find / -inst my_flp ]
```

(Default - *inherit*: The value is inherited from the root attribute.)

This is a context-based attribute and works on applies to both flops and latches based on the instance type.

- ◆ Messages

```
Info: Merging sequential instances that are equivalent. [GLO-42]
      : The instances are 'my_flop_0' and 'my_flop_1' in 'my_subd'
```

Sequential merging is power-domain aware, because no merging occurs across hierarchies and hence no merging is done across power domains.

# Multibit Cell Inferencing

---

You can group cells that are packaged into a single library cell with common pins by using:

```
set_attr use_multibit_cells {false | true}
```

- ◆ Multibit cells are more compact for place and route.
- ◆ Multibit cells have less power because of less capacitive loading.
- ◆ RC can map to the following multibit cells (when available in the library):
  - ❑ Sequential components such as latches and flip-flops (scan also)
  - ❑ Three-state cells
  - ❑ Single output combinational cells, such as MUXes, inverters, nand, nor, xor, and xnor.
- ◆ You can control multibit cell inferencing using pragmas.
- ◆ You can control naming styles to match verification tools.

04/25/11

Encounter RTL Compiler

135

Multiple choices of inferencing is based on:

- RTL pragmas or *infer\_multibit* attributes on elaborated design
- Automatic inferencing by module-level analysis to identify potential candidates for merging

Granularity of control:

- Design-level automatic inferencing
- Instance-level (OK, not OK)
- Ability to merge registers within and across register banks
- Ability to force merging of all registers to multibits
- Flexible naming styles for multibit instances

You can convert existing single-bit netlist to multibit.

Combinational cell multibit inferencing is not only limited to MUXes and three-states, but also any other single-output combinational cell, such as inverter, NAND, NOR, XOR, and XNOR.

Combinational cell multibit inferencing does not require specific annotation (although it can respect pragmas in RTL) and does not require user identification of candidates.

You can use the multibit mapping capability to map a group of flops to a register bank.

## Mapping to Complex Cells

---

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:

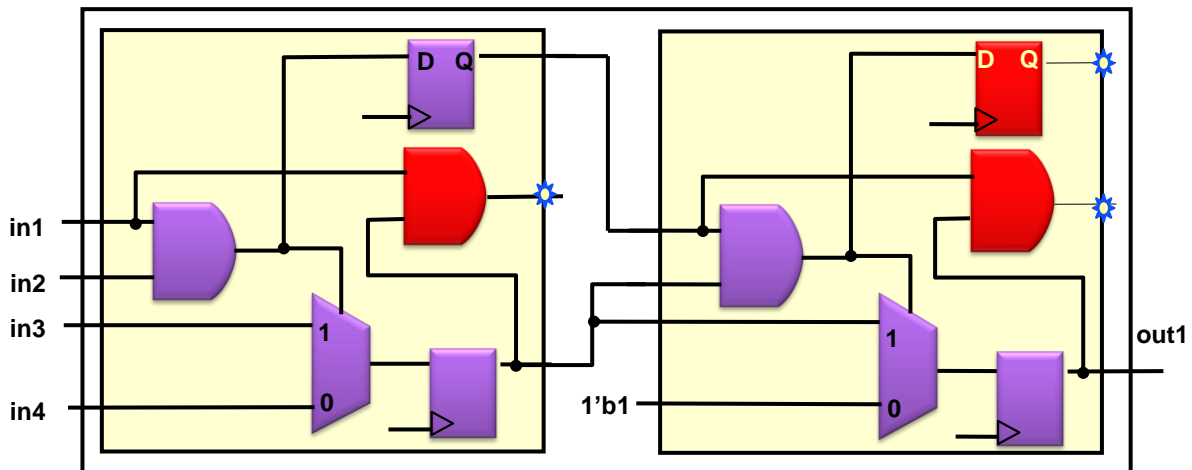
```
set_attr hdl_ff_keep_feedback true [find / -hdl_arch DFF*]
```

# 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>
```

In the example below, instances in red are deleted because they do not transitively drive an output port.



04/25/11

Encounter RTL Compiler

137

# Optimizing Logic Ending at Asynchronous Reset

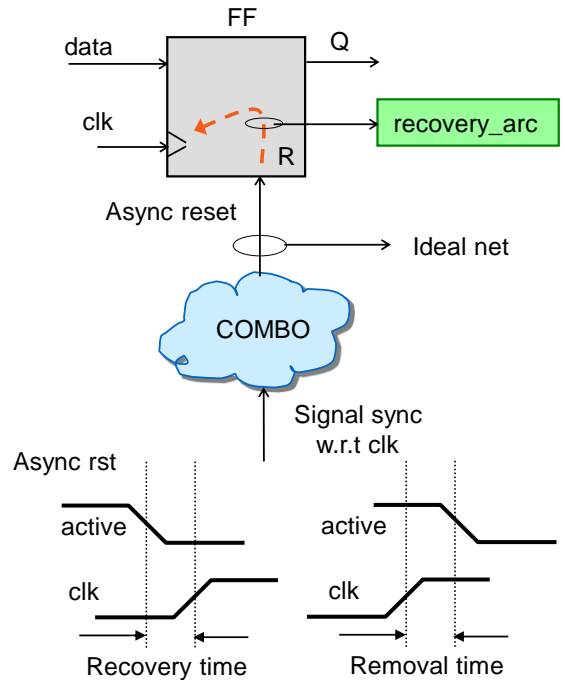
To optimize the combinational logic on the reset pin for timing, use the *recovery\_arc* in the library of the flop as a setup arc

```
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 <b>true</b>	attr <b>false</b>
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.**

04/25/11

Encounter RTL Compiler

139

## 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.

04/25/11

Encounter RTL Compiler

140

Encounter RTL Compiler will respect all preserved instances in the hierarchy.

You can also automatically ungroup user hierarchies during synthesis.

Use the *auto\_ungroup* attribute to specify whether RC should optimize for timing, area, or both timing and area during automatic ungrouping:

- `set_attribute auto_ungroup {none | timing | area | both}`

This attribute must be set before synthesis.

- In timing-driven ungrouping, RTL Compiler ungroups any timing critical blocks with 500 or less leaf instances.
- In area driven ungrouping, RTL Compiler ungroups all modules that have 300 or less leaf instances.

You can also specify the automatic ungrouping effort level. The higher the effort level, the more exhaustive the ungrouping will be at the cost of longer run time. The default value is high.

```
set_attribute auto_ungroup_min_effort {high | medium | low}
```



# Superthreading

---

You can use superthreading during generic synthesis, global mapping, and global incremental phases of the *synthesize* command.

To enable superthreaded optimization, set the attribute:

```
set_attribute super_thread_servers {machine_names} /
```

- ◆ The attribute value is a Tcl list representing the set of machines on which RTL Compiler can launch processes to superthread or batch for LSF and SGE.
- ◆ If you want to pass commands to the LSF or SGE queuing systems, use the *super\_thread\_batch\_command* and *super\_thread\_kill\_command* attributes. The *super\_thread\_batch\_command* attribute will pass commands to the queuing system when superthreading is launched, while the *super\_thread\_kill\_command* attribute will pass commands to remove jobs from the queuing system.

By default, the tool does superthreading automatically on multi-CPU servers to maximize the use of the multiple CPUs on the machine. This feature does not require an additional license.

You can turn automatic superthreading off using the following attribute:

```
set_attribute auto_super_thread false /
```

04/25/11

Encounter RTL Compiler

141

The following are the licensing requirements for superthreading:

- Main server that initiates superthreading requires an RC-GXL license.
- First remote server does not need a license.
- Other servers require at least an RC-L or an Encounter license.

LSF: Load sharing farm

SGE: Sun grid engine



## Lab 6-1 Exploring Optimization Strategies

- Setting PLE Mode and Optimization Attributes
- Running Generic Synthesis
- Mapping to a Target Library
- Running Incremental Synthesis
- Using Spatial Mode to Estimate Wire Delays
- Generating and Analyzing Reports
- Saving Synthesis Results

# Low-Power Synthesis

## Module 7

April 25, 2011



# Module Objectives

---

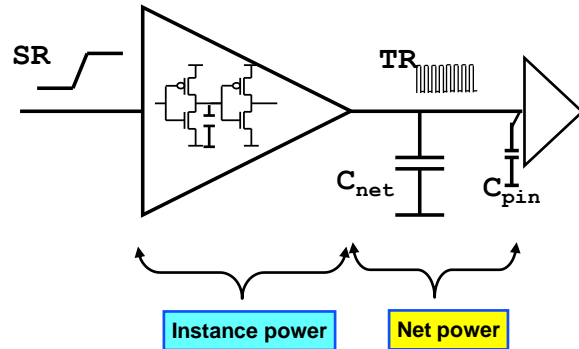
In this module, you

- ◆ Use the low-power synthesis flow
- ◆ Apply clock gating
- ◆ Annotate switching activity
- ◆ Analyze power consumption

# 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



04/25/11

Encounter RTL Compiler

145

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

# Motivation for Power Management

---

Longer Battery life between charges

Maximum power consumption is limiting performance

Cost and reliability of devices

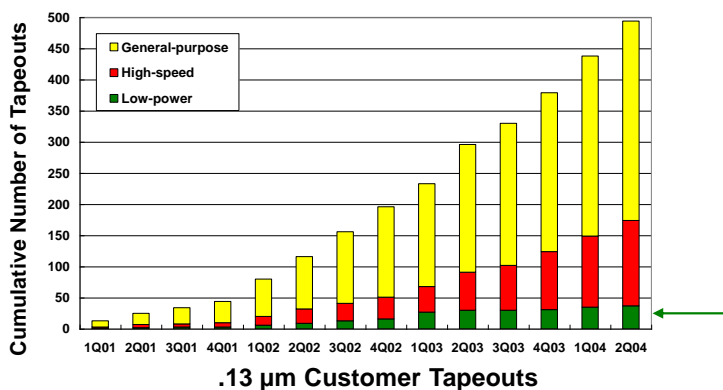


Everyone is motivated to do something about power management.

# Typical Product Tapeouts Trend

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.



A growing percentage of tapeouts use low power.

# Power Dissipation in CMOS

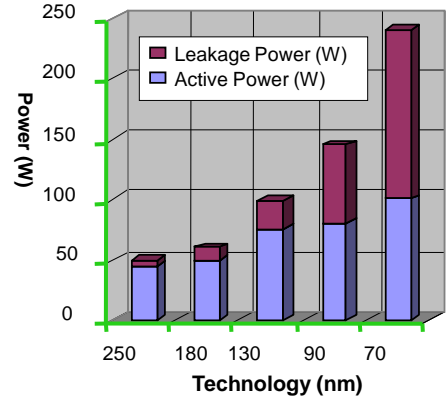
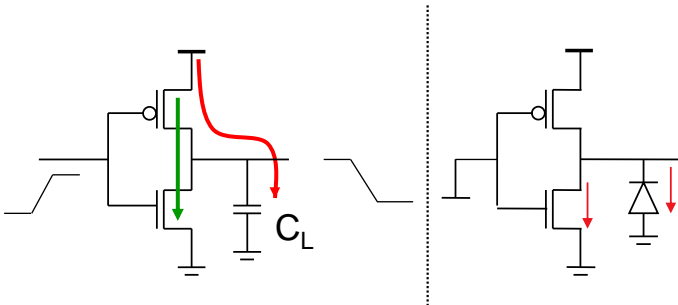
$$P_{\text{total}} = C_L V_{DD}^2 f_{0 \rightarrow 1} + t_{sc} V_{DD} I_{\text{peak}} f_{0 \rightarrow 1} + V_{DD} I_{\text{leakage}}$$

Dynamic

Short Circuit

Leakage

$$I_{\text{leakage}} = I_{\text{sub}} + I_{\text{gate oxide}} + I_{\text{junction}}$$



04/25/11

Encounter RTL Compiler

148

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 Vth library

- ◆ Timing compromise

### Reduce active power

- ◆ Dynamic optimization

- ◆ Timing compromise

- ◆ Clock gating

- ◆ Issues with *equivalence checking* (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.

# Power Reduction Technique Tradeoffs/Selection

Power reduction technique	Power Savings	Timing impact	Area impact	Methodology Impact			
				Design	Verification	CPF?	Licensing?
Area optimization	Small	-None-	n/a	Low	-None-	NO	RC-L, RC-XL
Multi-Vt optimization	Medium	Little	Little	Low	-None-	NO	RC-L, RC-XL
Clock gating	Medium	Little	Little	Low	Low	NO	RC-L, RC-XL
Multi-supply voltage (MSV)	Large	Some	Some	Medium	Medium	YES	RC-GXL
Power shut-off (PSO)	Large	Little	Some	High	High	YES	RC-L, RC-XL
Dynamic & Adaptive Voltage Frequency Scaling (DVFS)	Large	Some	Some	High	High	YES	RC-GXL

04/25/11

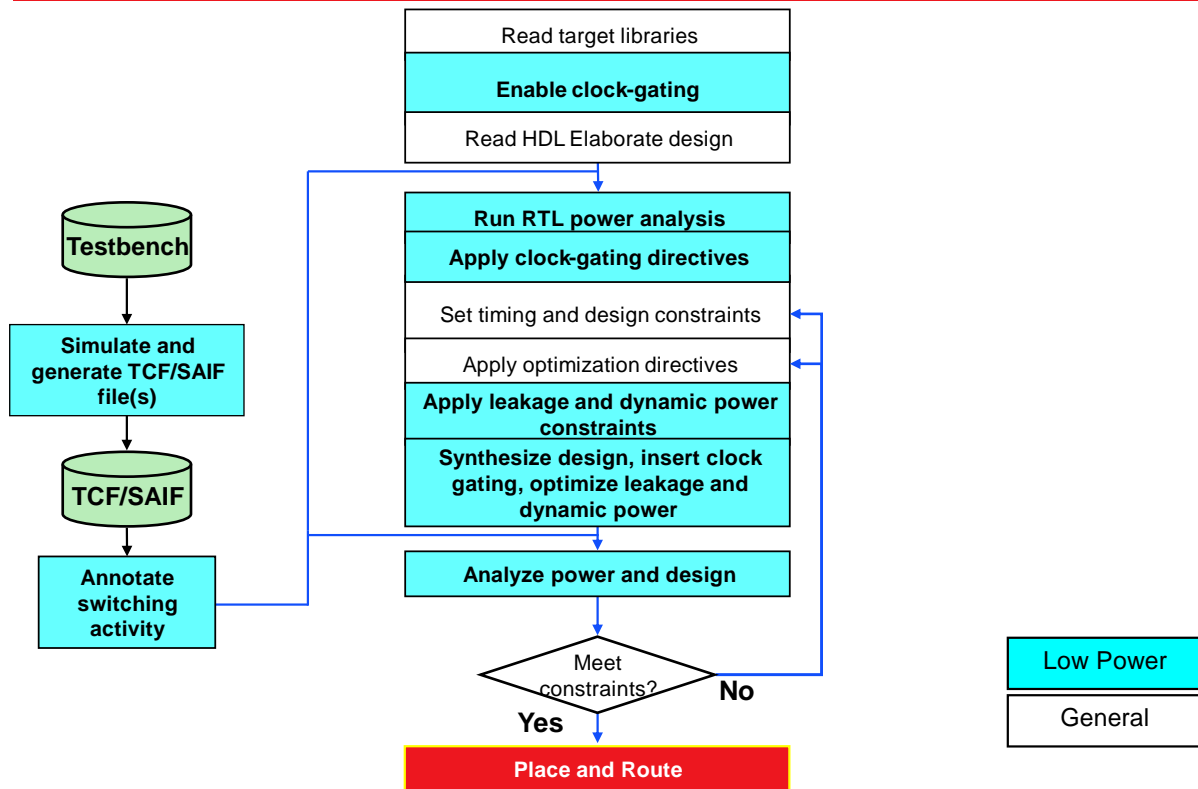
Encounter RTL Compiler

150

The synthesis techniques above have fewer tradeoffs but have less impact on power.

The big impact techniques have a big effect on timing, and even area. And the biggest impact is often on methodology, which translates to management as productivity and/or time-to-market. Thus they are often not adopted up-front in a project.

# Low-Power Only Flow



04/25/11

Encounter RTL Compiler

151

# 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:

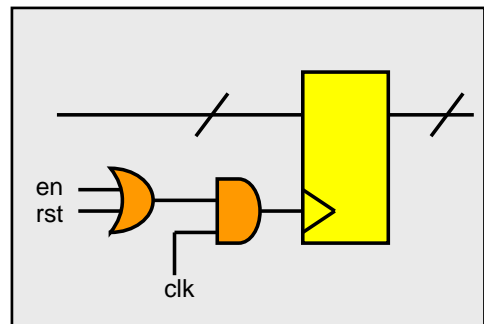
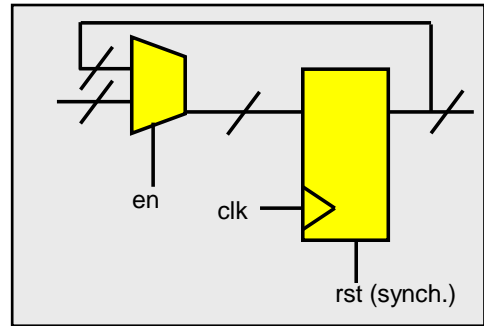
```
set_attr lp_insert_clock_gating true /
```

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: Infinite

To specify the minimum number of registers:

- ◆ `set_attr lp_clock_gating_min_flops integer /`
- ◆ Default: 3



04/25/11

Encounter RTL Compiler

152

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 Compiler will even “declone” gating logic with common enable signals across the hierarchy.

# Hierarchical Clock Gating

Hierarchical clock gating capability can handle various RTL code styles.

To turn on hierarchical clock gating, use:

```
set_attribute lp_clock_gating_hierarchical true /
```

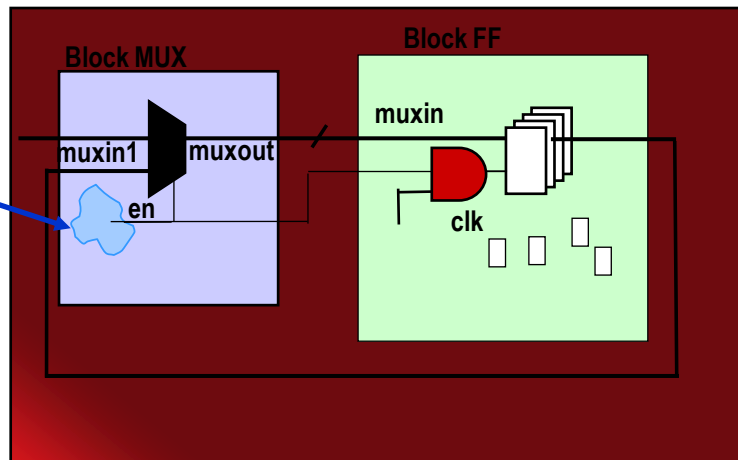
This example shows that the “if/case” logic and the registers can be defined in different modules.

## RTL

```
module mux(...)  
  always @(muxin0 or  
    muxin1 or en)  
  if(en)  
    muxout = muxin1;  
  else  
    muxout = muxin0;  
  _____  
  _____  
  _____
```

Control  
signal

```
module FF(...);  
  always@(posedge clk)  
    muxin <= muxout;  
  _____  
  _____  
  _____
```



# 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.

04/25/11

Encounter RTL Compiler

154

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>`

# 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:

```
build_rtl_power_models
report power
```

- ◆ RC reports the estimated power of your design, including:

- ☐ Multiple supply voltage domain power
- ☐ Clock-gating logic power

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

04/25/11

Encounter RTL Compiler

155

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

# Leakage Power Optimization

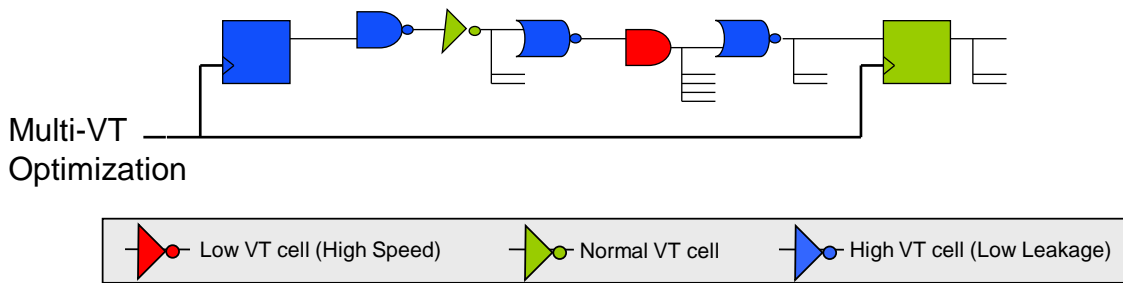
To enable leakage power optimization, after elaboration set the following power attribute:

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

To specify the power optimization effort, set the following power attribute:

```
set_attribute power_optimization_effort <low/med/high> /
```

- ◆ low : Reduces power without area impact.
- ◆ medium (default): Balances the area / power tradeoff.
- ◆ high: Further reduces power with area tradeoff.



04/25/11

Encounter RTL Compiler

156

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.

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.

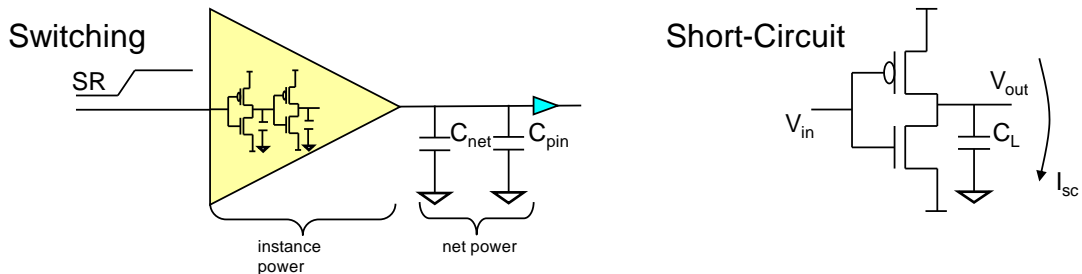


# Dynamic Power Optimization

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*/*
```



04/25/11

Encounter RTL Compiler

157

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
```

# 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.

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*.

# 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
=====
Operation              Worst
                        Total Neg
                        Area      Slack
-----
Path: coeff[3] --> regs/big_normal/data_out_reg[3]/D
power_map            2326      -158      7997      618910
...
Incremental optimization status
=====
Operation Area        Worst
                        Total - - DRC Totals - -
                        Total Neg Neg      Max      Max
                        Slack  Slack  Trans   Cap      Leakage   Switching
-----
...
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
```

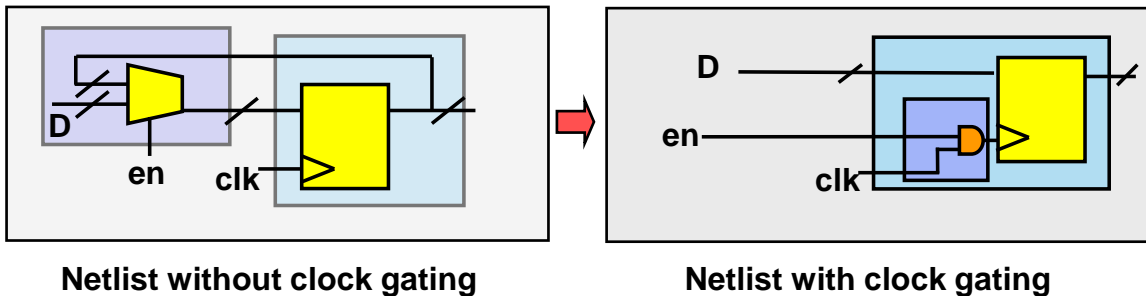
# Clock Gating: Insert Clock Gating in the Netlist

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]
```



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 Reports

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

```
report clock_gating      OR
report clock_gating -ungated_ff
```

## Summary

Category	Number	%
RC Clock Gating Instances	30	-
Non-RC Clock Gating Instances	0	-
RC Gated Flip-flops	468	92
Non-RC Gated Flip-flops	0	0
Ungated Flip-flops	43	8
Total Flip-flops	511	100

## Ungated Flip-flops

Flip-flop	Excluded	Module	Instance
dma_grant_reg	false	arb	dtmf_recvr_core/ARB_INST
present_state_reg[0]	false	arb	dtmf_recvr_core/ARB_INST
present_state_reg[1]	false	arb	dtmf_recvr_core/ARB_INST

# Clock Gating Reports (continued)

## report clock\_gating -detail

```
Detail
-----
Clock Gating Instance : RC_CG_HIER_INST2
-----
Origin:                Inserted by RC
Libcell:               TLATNTSCAX2M (ss_g_1v08_125c)
Style:                 latch_posedge_precontrol
Module:                dma (dtmf_recvr_core/DMA_INST)
Type:                  Leaf level CG Instance
Inputs:
  ck_in                =          clk (/designs/dtmf_recvr..L_INST/pins_out/m_clk)
                        TCF = (0.50000, 0.250000/ns)
  enable               =          n_9 (/designs/dtmf_recvr..comb/g528/pins_out/Y)
                        TCF = (0.07420, 0.005208/ns)
  test =               LOGIC0
Outputs:
  ck_out               =          rc_gclk
                        TCF = (0.02600, 0.020833/ns)
Gated FFs:
Module  Clock Gating Instance  Fanout  Gated Flipflops
-----
dma     RC_CG_HIER_INST2      7        a_reg[1]
                                   a_reg[2]
                                   a_reg[3]
                                   a_reg[4]
                                   a_reg[5]
                                   a_reg[6]
                                   a_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 -depth 1
```

Instance	Cells	Leakage Power (mW)	Dynamic Power (mW)	Total Power (mW)
dtmf_recvr_core	6412	0.045	14.036	14.081
TDSP_CORE_INST	4318	0.024	3.474	3.498
RESULTS_CONV_INST	1717	0.003	1.481	1.484

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*]
```

# Report Instance Power

## report instance -power

Instance Power Info				
-----				
Instance ROM_512x16_0_INST of Libcell rom_512x16A				
Leakage	Internal	Net		
Power (mW)	Power (mW)	Power (mW)		
-----				
0.017	5.789	0.001		
-----				
Pin	Net	Computed	Computed	Net
		Probability	Toggle Rate (/ns)	Power(mW)
-----				
Q[15]	rom_data[15]	0.5	0.0208	0.000
Q[14]	rom_data[14]	0.5	0.0208	0.000
...				
Arc	Arc	Arc	Arc	
From	To	When	Activity (/ns)	Energy (fJ)
-----				
CLK	CLK	!CEN	0.2500	23157.000
CLK	CLK	CEN	0.0000	10153.000



# Report Gates Power

## report gates -power

=====							
Gate	Instances	Area	Leakage Power (nW)	Internal Power (mW)	Library		
ADDFHX4M	31	1969.796	0.002	0.015	ss_g_1v08_125c		
ADDFHX8M	10	788.389	0.001	0.004	ss_g_1v08_125c		
rom_512x16A	1	41815.630	0.017	5.789	rom_512x16A_slow		
-----							
total	6467	230202.134	0.045	9.717			
=====							
Library	Instances	Instances %	Area	Leakage Power (mW)	Leakage Power %	Internal Power (mW)	Internal Power %
-----							
pllclk	1	0.0	230202.134	0.000	0.0	0.000	0.0
ram_256x16_slow	2	0.0	230202.134	0.000	0.0	2.873	29.6
rom_512x16A_slow	1	0.0	230202.134	0.017	37.4	5.789	59.6
ss_g_1v08_125c	1287	19.9	230202.134	0.022	47.3	0.248	2.5
ss_hvt_1v08_125c	5176	80.0	230202.134	0.007	15.2	0.808	8.3
-----							
Type	Instances	Area	Area %	Leakage Power (mW)	Leakage Power %	Internal Power (mW)	Internal Power %
-----							
timing_model	4	160736.826	69.8	0.019	40.9	8.662	89.1
sequential	541	17109.218	7.4	0.003	6.6	0.588	6.0
inverter	674	3426.550	1.5	0.002	4.3	0.041	0.4
buffer	104	851.931	0.4	0.001	2.1	0.027	0.3
logic	5144	48077.609	20.9	0.023	49.6	0.400	4.1
-----							
total	6467	230202.134	100.0	0.045	103.5	9.717	100.0

04/25/11

Encounter RTL Compiler

165

## 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.

# write\_template -split -power -outfile run.tcl

## setup\_run.tcl:

```
## Power root attributes
set_attributelp_insert_clock_gatingtrue /
#set_attributelp_clock_gating_prefix<string> /
#set_attributelp_power_analysis_effort<high> /
#set_attributelp_power_unitmW /
#set_attributelp_toggle_rate_unit/ns /
## Uncomment to turn off hierarchical clockgating
#set_attributelp_clock_gating_hierarchicalfalse /
set_attributehdl_track_filename_row_coltrue /
set_attrpower_optimization_effortlow /
```

Enable clock-gating

## power\_run.tcl:

```
#####
#          LOW POWER setup (Leakage/Dynamic power/Clock Gating setup)          #
#####
#####
## Leakage/Dynamic power/Clock Gating setup.
#####
#set_attributelp_clock_gating_cell[find /lib*-libcell <cg_libcell_name>] "/designs/$DESIGN"
set_attributemax_leakage_power0.0 "/designs/$DESIGN"
#set_attributelp_power_optimization_weight<value from 0 to 1> "/designs/$DESIGN"
#set_attributemax_dynamic_power<number> "/designs/$DESIGN"
#set_attributelp_clock_gating_test_signaltest_signal_object "/designs/$DESIGN"
## read_tcf <TCF file name>
## read_saif <SAIF file name>
## read_vcd <VCD file name>
```

Configure clockgating attributes

Set leakage and dynamic power constraints to use during mapping.

Simulate the design and read the probability data.

# run.tcl for write\_template-split -power

---

```
#####
## Synthesizing to generic
#####

...

#### Build RTL power models
##build_rtl_power_models-design $DESIGN -clean_up_netlist[-clock_gating_logid \
    [-relative <hierarchical instance>]
#report power-rtl
...

#####
## write Encounter file set (verilog, SDC, config, etc.)
#####

...

report clock_gating> $_REPORTS_PATH/${DESIGN}_clockgating.rpt
report power -depth 0 > $_REPORTS_PATH/${DESIGN}_power.rpt
report gates-power > $_REPORTS_PATH/${DESIGN}_gates_power.rpt
...
```

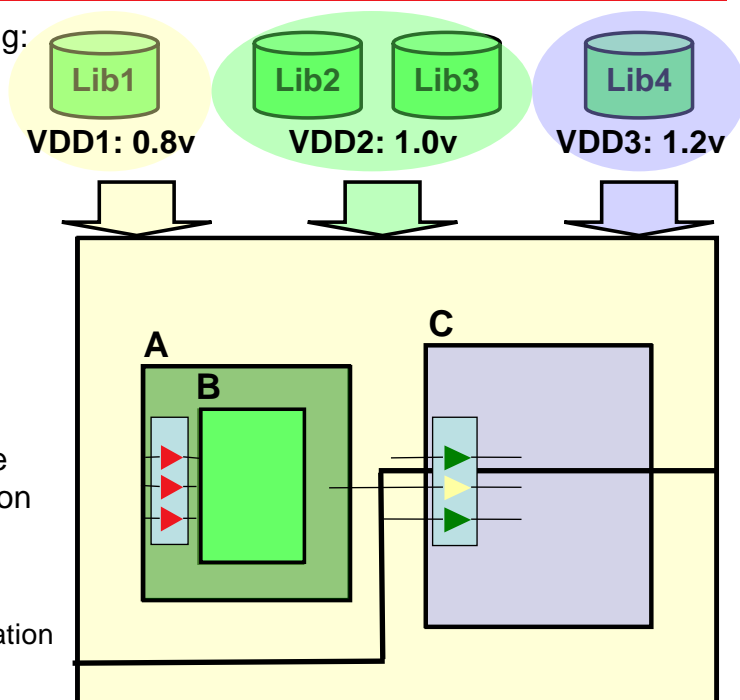
RTL Power  
Estimation

Analyze power  
results

# Top-down MSV Synthesis

MSV features include the following:

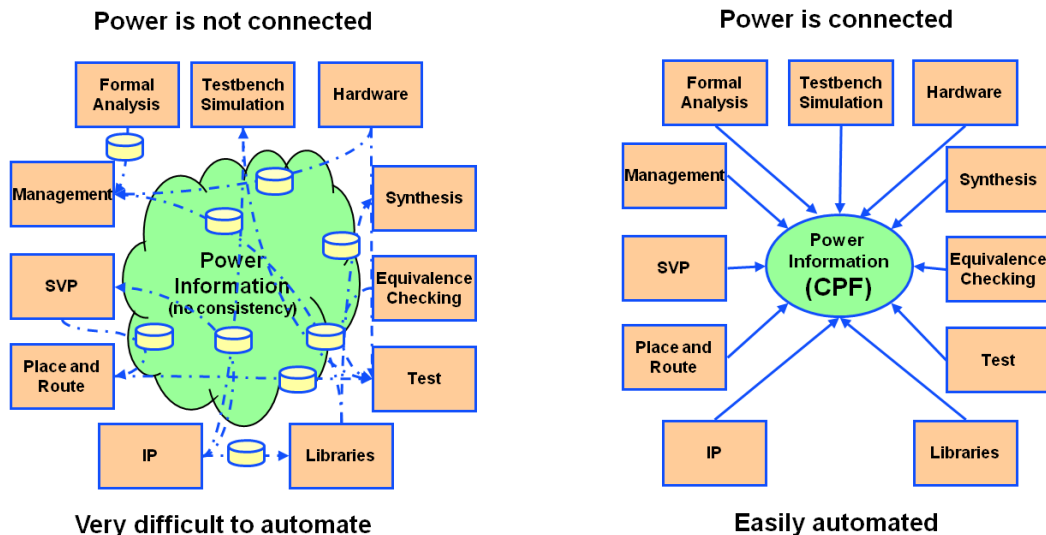
- ◆ Multiple voltage domains
  - ❑ Assign libraries to domains
  - ❑ Assign blocks to domains
- ◆ Top-down analysis and optimization
- ◆ Incremental what-if analysis
- ◆ Level shifter insertion
- ◆ Automatic transfer of voltage domain and shifter information to back-end tools
  - ❑ Place and Route
 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



# 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 Golden CPF can be used throughout the flow by all the tools.



04/25/11

Encounter RTL Compiler

170

## CPF Characteristics

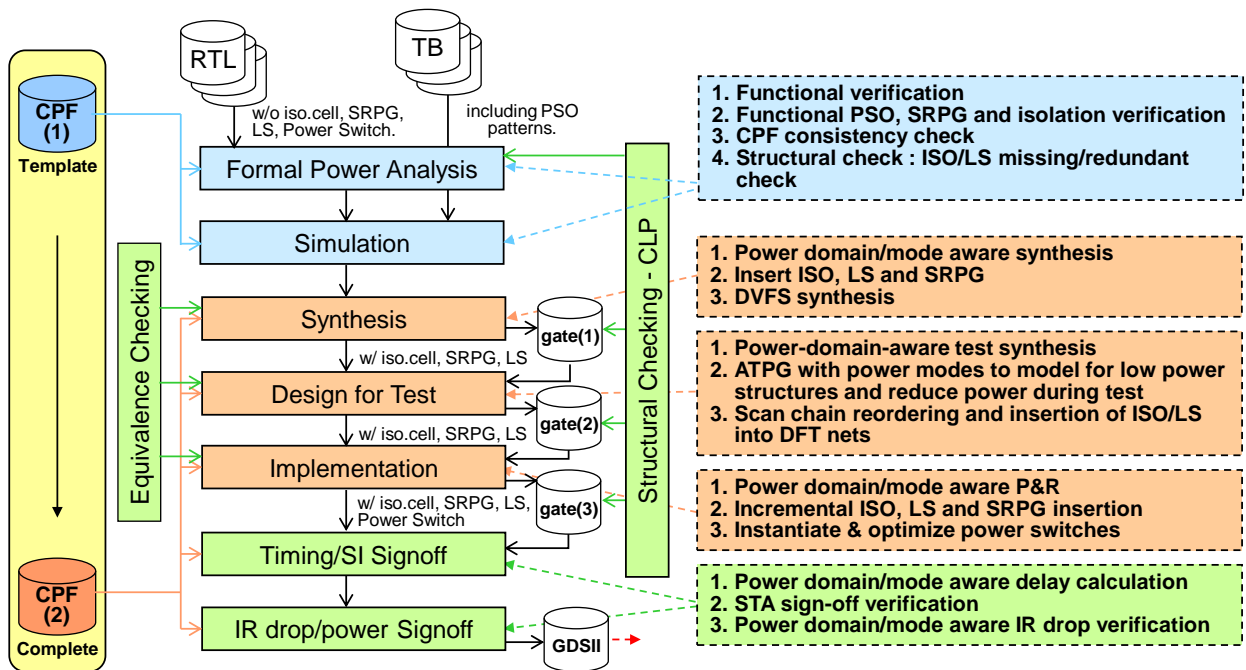
- CPF is TCL-based.
- CPF language = TCL commands + CPF objects + Design objects

## 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



04/25/11

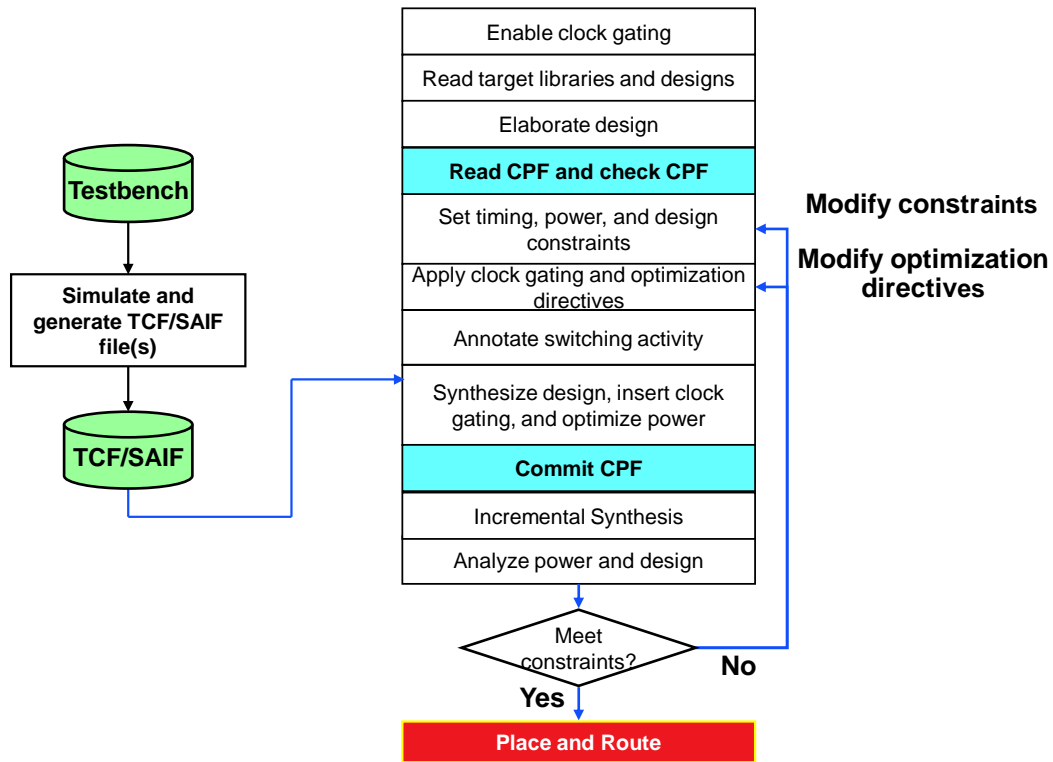
Encounter RTL Compiler

171

DVFS: Dynamic Voltage Frequency Scaling

ATPG: Automatic Test Pattern Generation

# Low-Power Flow Using Common Power Format



04/25/11

Encounter RTL Compiler

172

CPF: Common Power Format



# Example Design

## Three domains

- ◆ Top
- ◆ Lower
- ◆ Shutoff

## State retention in Top/Shutoff/B

## Isolation on outputs of shutoff

- ◆ Mix of low and high

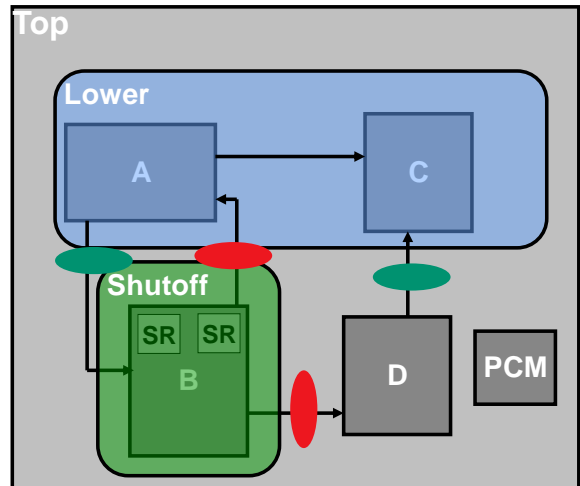
## Level shifters

## Lower and Shutoff can operate at two voltages

## Power modes

- ◆ All on
- ◆ Shutoff off

## Power Control Module controls power switching



# Domain Definition

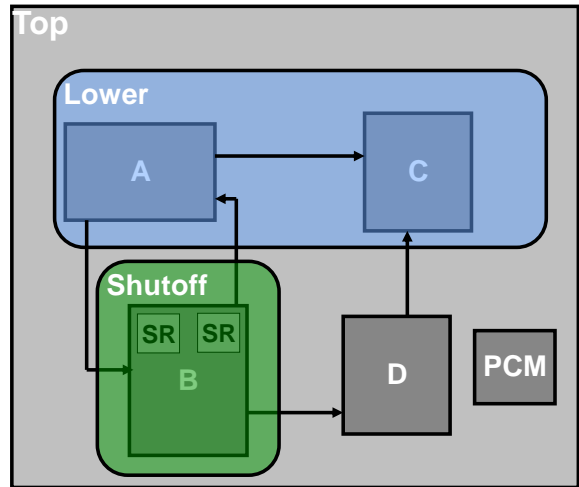
```
# Define the top domain
set_design TOP

# Define hierarchy separator
set_hierarchy_separator "/"

# Define the default domain
create_power_domain -name pdTop \
    -default

# Define Lower
create_power_domain -name pdLower \
    -instances {uA uC}

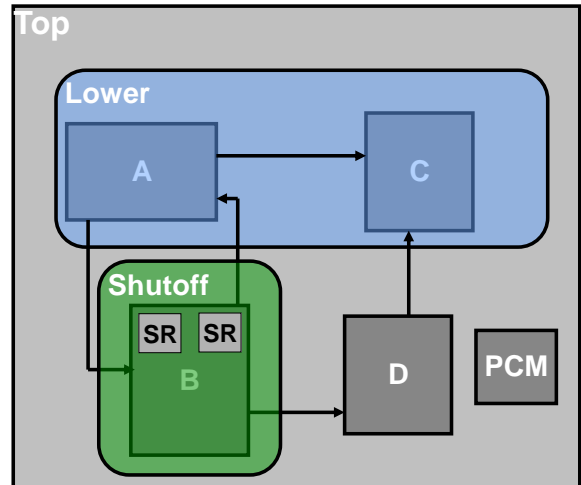
# Define Shutoff.  Shuts off when
# pso is low
create_power_domain -name pdShutoff \
    -instances {uB} \
    -shutoff_condition {!uPCM/pso}
```



# State Retention

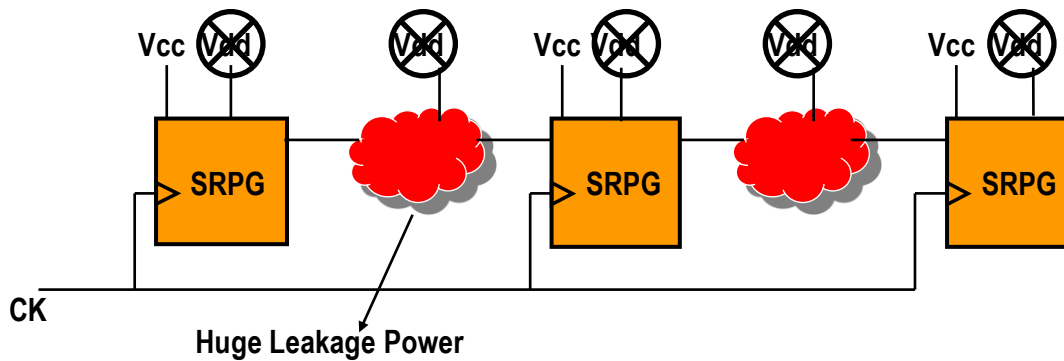
```
# Define SRPG
set srpgList {uB/reg1 uB/reg2}

create_state_retention_rule \
  -name srl \
  -restore_edge {uPCM/restore[0]} \
  -instances $srpgList
```



## State Retention Power Gating (SRPG)

- ◆ Even with clk off in sleep mode, leakage power still large at  $\leq 90\text{nm}$
- ◆ SRPG can reduce leakage power 10x
- ◆ Additional power gating pins added to flip-flop

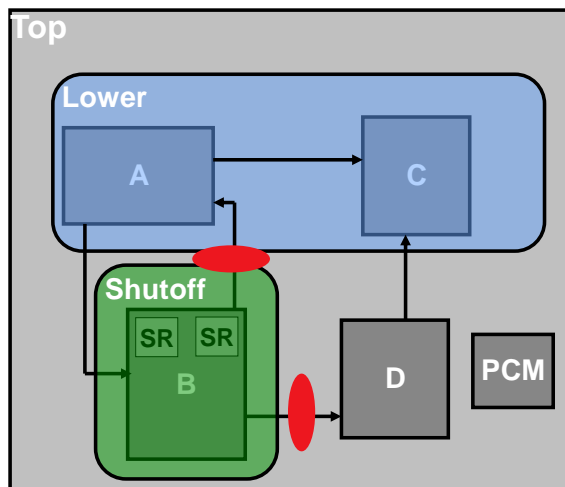



# Isolation

```
# Active high iso
set hiPin {uB/en1 uB/en2}

create_isolation_rule -name ir1 \
  -from pdShutoff \
  -isolation_condition {uPCM/iso} \
  -isolation_output high \
  -pins $hiPin

# Default rule (active low)
create_isolation_rule -name ir2 \
  -from pdShutoff \
  -isolation_condition {uPCM/iso} \
  -exclude $hiPin
```



 **Isolation cell  
(LS to come)**

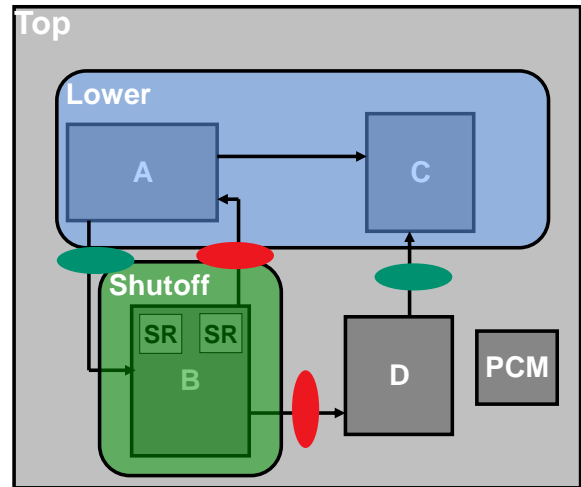
# Level Shifter

```
# Define the LS in the "to" domain
create_level_shifter_rule -name lsr1 \
  -to {pdShutoff} \
  -from {pdLower}

create_level_shifter_rule -name lsr2 \
  -to {pdLower} \
  -from {pdShutoff}

create_level_shifter_rule -name lsr3 \
  -to {pdTop} \
  -from {pdShutoff}

create_level_shifter_rule -name lsr4 \
  -to {pdLower} \
  -from {pdTop}
```



 **Isolation cell  
+ level shifter**       **level shifter**

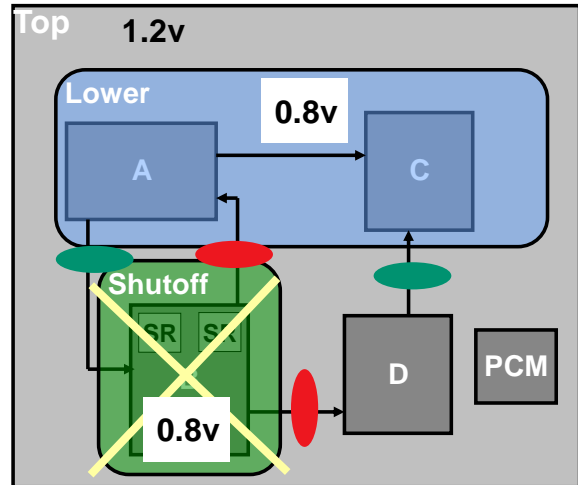
# Power Modes

```
# Define the conditions. Top is
# always high. Lower/Shutoff can be
# medium or low
create_nominal_condition -name \
    high -voltage 1.2
create_nominal_condition -name \
    medium -voltage 1.0
create_nominal_condition -name low \
    -voltage 0.8
create_nominal_condition -name off \
    -voltage 0

# Define the modes
create_power_mode -name PM1 \
    -domain_conditions {pdTop@high \
        pdLower@medium pdShutoff@medium}
create_power_mode -name PM2 \
    -domain_conditions {pdTop@high \
        pdLower@low pdShutoff@low}

# Mode where Shutoff is off
create_power_mode -name PM3 \
    -domain_conditions {pdTop@high \
        pdLower@low pdShutoff@off}

# End the design (for completeness)
end_design
```

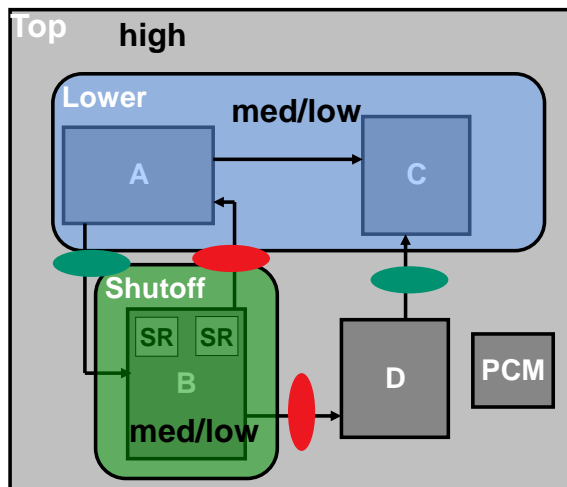


# Assign Libraries

```
# Define the libraries
define_library_set -name HIGH_set \
  -libraries {high1.lib high2.lib}
define_library_set -name MED_set \
  -libraries {med1.lib med2.lib}
define_library_set -name LOW_set \
  -libraries {low1.lib low2.lib}

# Associate the library sets with the
# operating condition (defined with
# create_nominal_condition previously)
update_nominal_condition -name high \
  -library_set HIGH_set
update_nominal_condition -name medium \
  -library_set MED_set
update_nominal_condition -name low \
  -library_set LOW_set

# Assign libraries to power domains
update_power_domain -name pdTop \
  -library_set HIGH_set
update_power_domain -name pdLower \
  -library_set LOW_set
update_power_domain -name pdShutoff \
  -library_set LOW_set
```



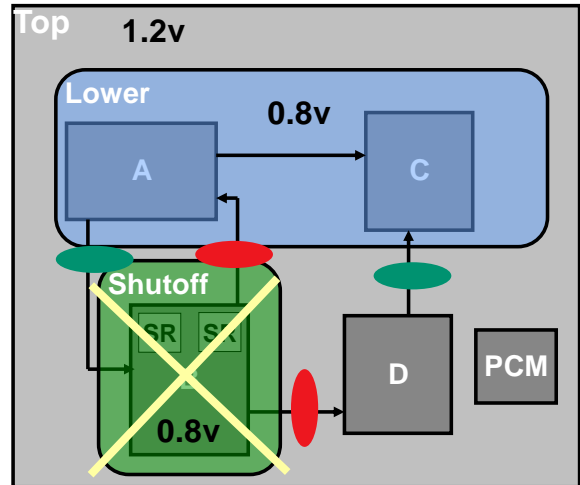


# Update Low Power Design Rules

```
# Update state retention rules
update_state_retention_rules \
  -name srl \
  -cell_type DRFF \
  -library_set MED_set
```

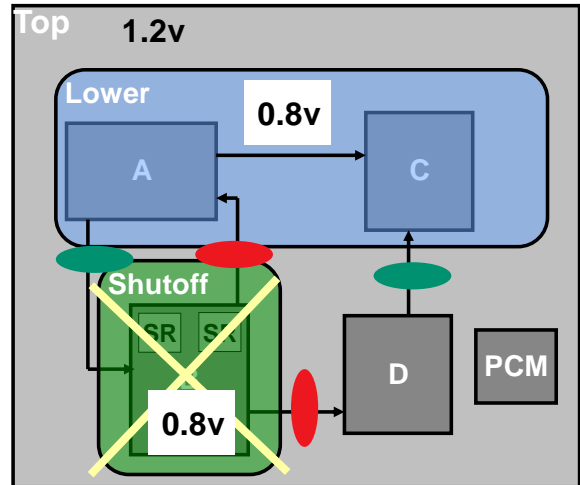
```
# Update isolation rules
update_isolation_rules \
  -name ir1 \
  -cells iso1 -location to
update_isolation_rules \
  -name ir2 \
  -cells iso2 -location to
```

```
# Update level shifter rules
update_level_shifter_rules \
  -name ls1 \
  -cells lvl -location to
```



# Define Multiple Modes

```
# Associate the sdc mode files
with
# the power modes (defined with
# create_power_mode previously)
update_power_mode -name PM1 \
    -sdc_files power_model1.sdc
update_power_mode -name PM2 \
    -sdc_files power_model2.sdc
update_power_mode -name PM3 \
    -sdc_files power_model3.sdc
```



# Check Library for Low-Power Cells

---

Library Domain	Total cells	LS cell	ISO cell	Combo (LS+ISO)	SR Flops
ao_wc_1v08(1.08)	405	1	0	0	16
tdsp_wc_0v84(0.84)	400	0	0	0	12
Unusable libcells					
Library Domain	Total cells	LS cell	ISO cell	Combo (LS+ISO)	SR Flops
ao_wc_1v08(1.08)	166	16	0	4	0
tdsp_wc_0v84(0.84)	36	12	0	4	0

# Report Power Domains

## Summary

=====

		Shut-off	signal	
Name		Name	Active level	Voltage (V)
AO (*)		-	-	1.08
PLL		-	-	1.08
TDSPCore	PM_INST/pd_inst/power_state_i/power_switch_enable		active_high	0.84

# write\_template -split -cpf -power -outfile run.tcl

```
#####
## Template CPF file
#####
set_cpf_version 1.1
set_hierarchy_separator <hierarchy separator>

### Technology part of the CPF

define_library_set_name <name> -libraries <library list>

#### Level Shifters

define_level_shifter_cell -cells <cell name> \
    -input_voltage_range <value> \
    -output_voltage_range <value> \
    -direction <value> \
    -input_power_pin <pin> \
    -output_power_pin <pin> \
    -ground <pin> \
    -valid_location <location>

define_isolation_cell -cells <cell name> \
    -valid_location <location> \
    -power_switchable \
    -ground_switchable \
    -enable <pin>

#####
## Headers
#####
define_power_switch_cell -cells <cell name> \
    -power_switchable <value> -power <value> \
    -stage_1_enable <value> \
    -stage_1_output <value> \
    -type <value>
```

```
#####
### State retention cells
#####
define_state_retention_cell -cells <cell names> \
    -clock_pin <pin> \
    -power <value> \
    -power_switchable <value> \
    -ground <value> \
    -save_function <value> \
    -restore_function <value>

#####
## Always ON Cells
#####
define_always_on_cell -cells <cell list> \
    -library_set <library set> \
    -power_switchable <LEF_power_pin> (or) -ground_switchable
    <LEF_ground_pin> \
    -power <LEF_power_pin> \
    -ground <LEF_ground_pin>

#### Design part of the CPF

set_design <design name>
create_power_nets -nets <name> -voltage <value>
create_ground_nets -nets <name>

### Create power domains

create_power_domain -name <domain name> -default
create_power_domain -name <domain name> -instances <instance list>

create_global_connection -domain <domain name> -net <net name> -pin <pin
name>

update_power_domain -name <name> -internal_power_net <net name>

create_nominal_condition -name <name> -voltage <value>
update_nominal_condition -name <name> -library_set <library set name>

create_power_mode -name <name> -domain_conditions <conditions> -default
```

What's New?  
The template.cpf file.

04/25/11

Er

## template.cpf (continued)

---

```
update_power_mode-name <name>-sdc_files<sdc file_name>
create_state_retention_rule-name <name>\
    -domain <domain name>\
    -restore_edge<value>\
    -save_edge<value>
update_state_retention_rules-names <name>\
    -cell_type<value>\
    -library_set<value>
create_level_shifter_rule-name <name>-from <domain>-to <domain>
update_level_shifter_rules-names <name>-cells <cell name>-location <location value>-prefix <name>
create_isolation_rule-name <name>\
    -isolation_condition<pin>\
    -pins -from <power_domain> \
    -to <power_domain> \
    -isolation_output<low/high> \
    -isolation_target<from|to>
update_isolation_rules-name <name>-location <location value>-prefix <name>
create_power_switch_rule-name <rule name>-domain <domain name>\
    -external_power_net<net name>
update_power_switch_rule-name <rule name>\
    -cells <name>\
    -prefix <value>
end_design
```

# run.tcl for Low Power and CPF-based Flow

---

What's New: setup\_run.tcl:

```
read_cpf -library run.cpf
```

What's New: run.tcl:

```
#### Read in CPF file.
```

```
read_cpf template.cpf
```

```
include power_run.tcl
```

```
check_cpf
```

```
## Reapply CPF rules
```

```
## reload_cpf -design /designs/$DESIGN
```

```
commit_cpf
```

```
##write_encounterdesign -basename <path & base filename> -lef <lef_file(s)>
```

```
verify_power_structure-post_synth > $_REPORTS_PATH/${DESIGN}_verify_power_struct.rpt
```

# RTL-based Power Versus Timing Design Exploration

Rapidly explore multi-supply multi-voltage possibilities

- ◆ Report performance, power, area for each combination

Built natively into synthesis engine

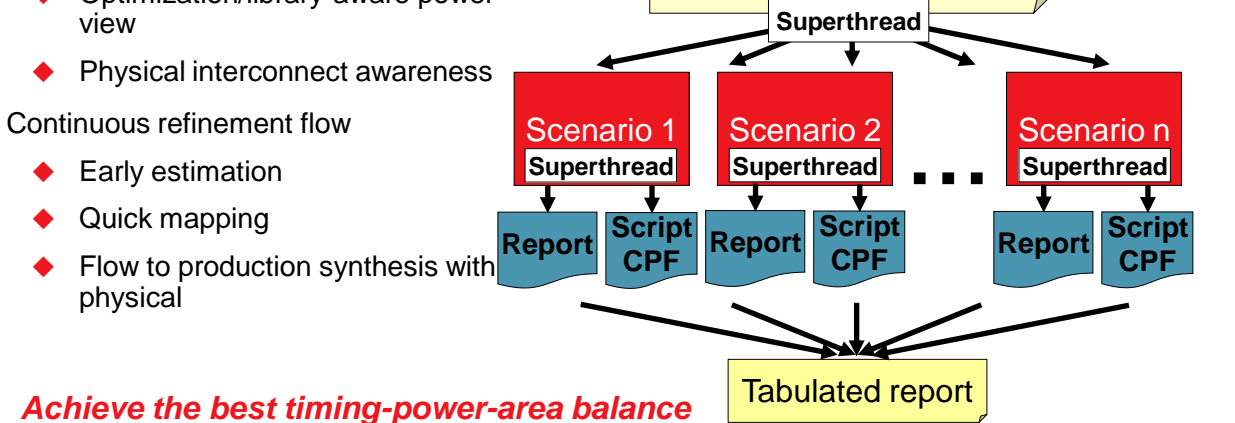
- ◆ Timing view
- ◆ Optimization/library-aware power view
- ◆ Physical interconnect awareness

Continuous refinement flow

- ◆ Early estimation
- ◆ Quick mapping
- ◆ Flow to production synthesis with physical

- RTL
- Target libraries
- Timing constraints
- CPF (library info)
- Exploration setting
- Switching activity
- Synthesis settings

- ☐ Power domains
- ☐ Voltage ranges
- ☐ Level shifter rules



04/25/11

Encounter RTL Compiler

188

We have built RTL exploration into RTL Compiler to help explore MSMV synthesis approaches. You pass your normal synthesis constraints, libraries, RTL, and scripts, and you specify exploration parameters. It then launches exploration runs in parallel. These exploration runs take advantage of RC global synthesis to quickly report back accurate performance, power, and area numbers for each possible scenario, in a tabulated report. It can even take advantage of physical interconnect modeling in RC for even more accuracy.

When you have chosen a scenario to move forward, RC generates the scripts and CPF to run it through full production synthesis.





# Lab Exercises

---

## Lab 7-1 Running Low-Power Synthesis

- Enabling Clock Gating and Operand Isolation
- Setting Up for Low-Power Synthesis
- Annotating Switching Activity
- Running Low-Power Synthesis
- Reporting Power

---

Blank Page

# Interface to Other Tools

## Module 8



April 25, 2011

# 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:

**bitblast\_all\_ports:** Bitblasts all ports of a design or subdesign

**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] ;
  ...
```

04/25/11

Encounter RTL Compiler

194

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	default
root	write_vlog_top_module_first	false
root	write_vlog_unconnected_port_style	full

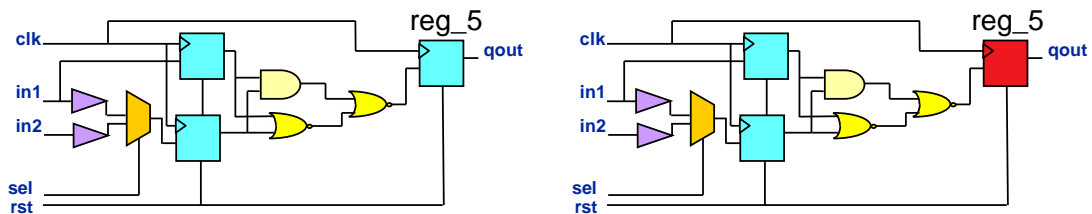
# Changing the Instance Library Cell

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
```



# Remove Assigns

---

You can remove the assign statements from your design by setting the *remove\_assigns* attribute to true.

```
set_attr remove_assigns true /
```

- ◆ You can also add certain specific options to the assign removal with the following command:

```
set_remove_assign_options [options] -design [subd |  
design]
```

- ◆ The attribute must be set before running synthesis on your design so that incremental optimization can work on the added buffers, or you will have to run an additional incremental optimization stage.

To remove assigns without performing any optimization on the added buffers, use:

```
remove_assigns_without_optimization [options]
```



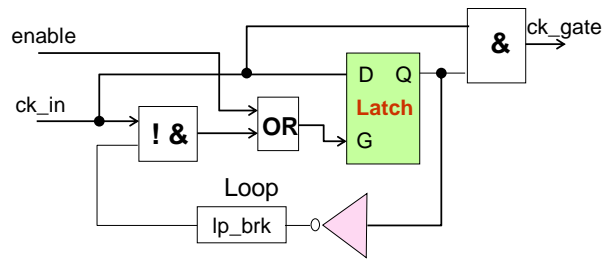
# Removing the Loop Breaker Cells

RC inserts the *cdn\_loop\_breaker* instances to break combinational feedback loops. To remove these instances from the netlist, use the following commands:

```
report cdn_loop_breaker -sdcfile $DESIGN.sdc
remove_cdn_loop_breaker -instances [instance_list] des
write_hdl > ${DESIGN}_netlist.v
```

- ◆ The *-sdcfile* option creates an SDC file with the appropriate *set\_disable\_timing* settings.

```
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
```



04/25/11

Encounter RTL Compiler

197

RTL Compiler automatically analyzes the elaborated design for combinational feedback loops during timing analysis; for example, when using the *report timing* or the *synthesize* command. Upon detecting such loops, and before performing timing analysis, RTL Compiler selects a buffer from the technology library to serve as a loop breaker instance. RTL Compiler then instantiates this cell along the feedback loop and disables the timing arc inside the cell.

These cells follow the *cdn\_loop\_breaker<number>* nomenclature, and therefore, they are easily identifiable in the netlist using the *find* command.

For example:

```
rc:/> find /designs -instance cdn_loop_breaker*
```

# 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
```

# 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_

04/25/11

Encounter RTL Compiler

200

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 <Encounter Configuration File>
```

## Example

```
read_encounter config design.conf
```

## Syntax

```
write_encounter design <string> -basename <string> [-reference_config_file  
<string>] [-ignore_scan_chains] [-ignore_msv] [-gzip_files]  
[-reference_config_file <string>]: file to use as a template  
[-lef <string>]: specify or override physical library file(s)
```

## Examples

```
write_encounter design top -basename FINAL/final
```

```
Unix> ls FINAL
```

```
final.conf final.enc_setup.tcl final.mode final.sdc final.v
```

```
write_encounter design top -gzip_files
```

```
Unix> ls rc_enc_des
```

```
rc.conf rc.def rc.enc_setup.tcl rc.mode rc.sdc.gz rc.v.gz
```

04/25/11

Encounter RTL Compiler

202

To use a specific release of RC with First Encounter<sup>®</sup> software, do the following:

- 1 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
```

- 2 Modify your PATH environmental variable to contain this path.

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

- 3 Set the CDS\_USE\_PATH variable to 1.

```
setenv CDS_USE_PATH 1
```

- 4 Start the First Encounter software. Before using the *runN2N* command, run the *which rc* command at the prompt. It displays *~/myBin/rc*.

Now you can use the *runN2N* command with 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® Equivalence Checker 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]
```

04/25/11

Encounter RTL Compiler

203

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
```

# Interface to Conformal Constraint Designer

---

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

```
write_do_ccd {generate | propagate | validate}...
```

- ◆ **generate:** Generates a dofile for the *Generate* flow, which generates additional false paths based on critical path timing reports.
- ◆ **propagate:** Generates a dofile to create a chip-level SDC file by propagating block-level constraints to the top-level and by integrating them with the glue constraints.
- ◆ **validate:** Generates a dofile for the *Validate* flow, which validates the constraints and false path exceptions.

You can also verify any issues with your clock domain crossings using the validate flow template and add in the clock domain crossing settings.



## Other RC Interfaces

---

RTL Compiler interfaces with a lot of other Encounter software to simplify your design process.

- ◆ Use the following command to interface with the Encounter Conformal Low-Power software:

```
write_do_clp
```

- ◆ Use the following commands to interface with the Encounter Test software:

```
write_et_atpg, write_et_bsv, write_et_rrfa, write_et_mbist
```

- ◆ Use the following command to interface with the Encounter Timing System software:

```
write_ets
```

# Lab Exercises

---

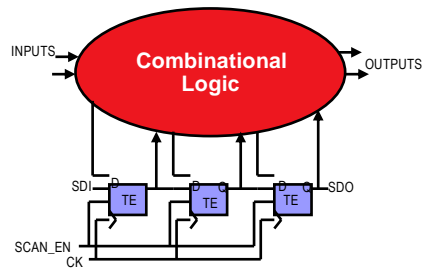


## Lab 8-1 Interfacing with Other Tools

- 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

# Test Synthesis

## Module 9



April 25, 2011

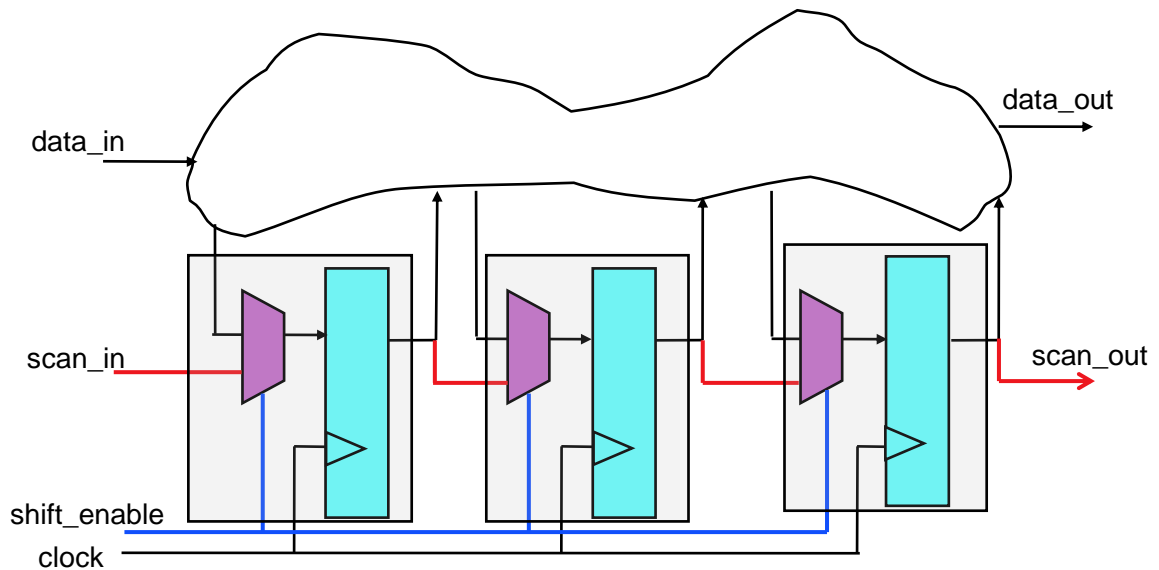
# Module Objectives

---

In this module, 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



04/25/11

Encounter RTL Compiler

209

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.

# What Is Your Test Plan?

---

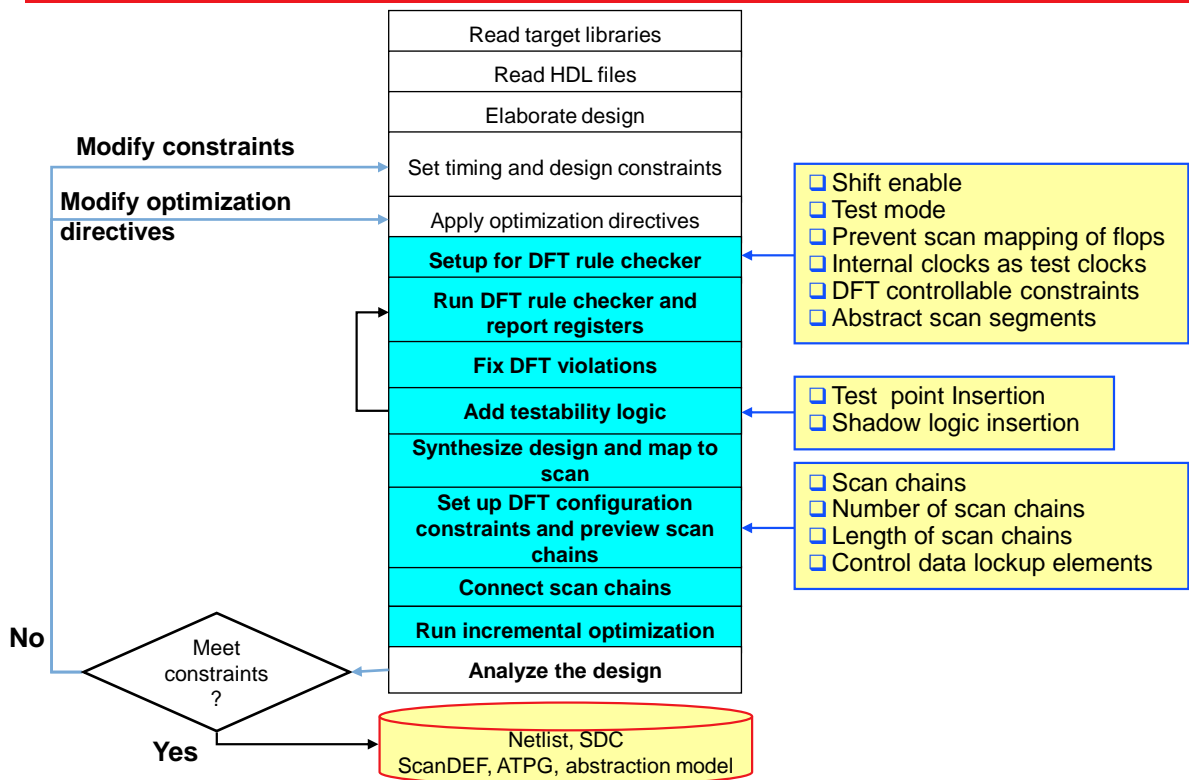
RC is extremely flexible with it's DFT implementations

You must know what your DFT requirements are before you try to implement them in RC

RC can easily hookup scan to existing 3rd Party Test DFT logic structures

And it can provide a broad range of DFT analysis & test logic insertion capabilities

# RTL Top-down Design-for-Testability (DFT) Flow



04/25/11

Encounter RTL Compiler

211

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:

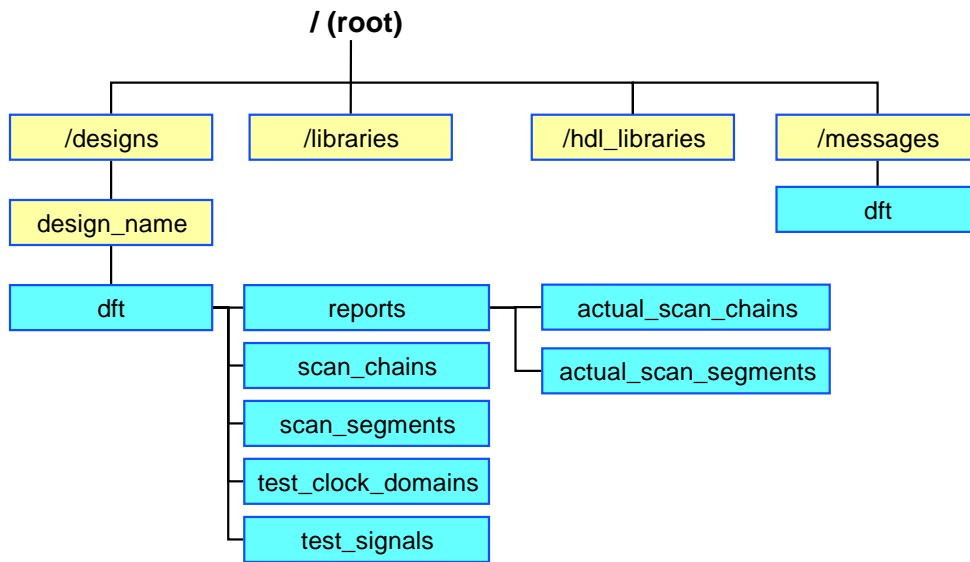
- Scan insertion
- Memory BIST insertion
- Logic BIST insertion
- Boundary scan insertion

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

---





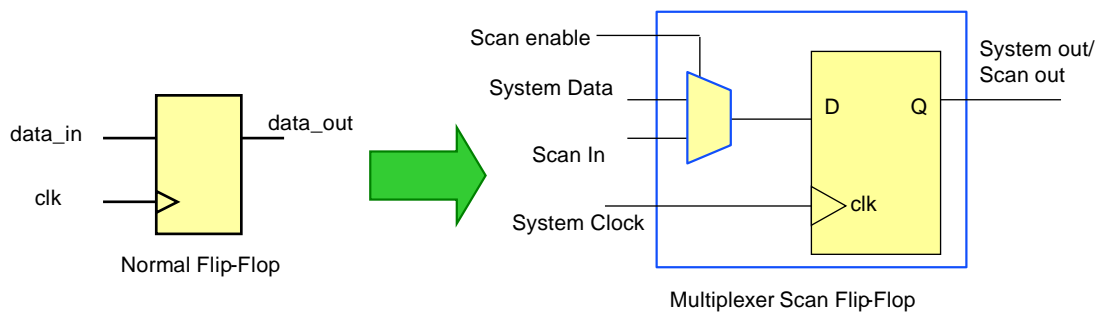
# Scan Styles: MUX Scan

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 rootlevel attribute:

```
set_attr dft_scan_style muxed_scan /
```

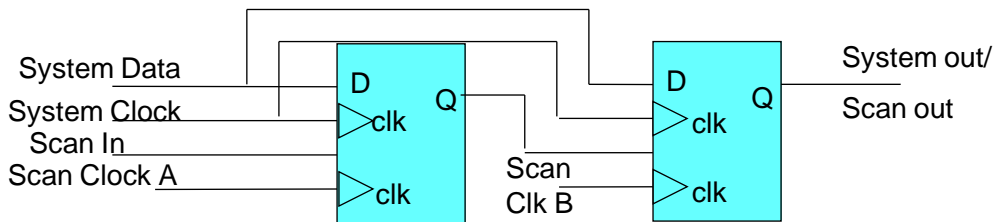


# Scan Styles: Clocked LSSD

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 /
```



04/25/11

Encounter RTL Compiler

214

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 without 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}]
```

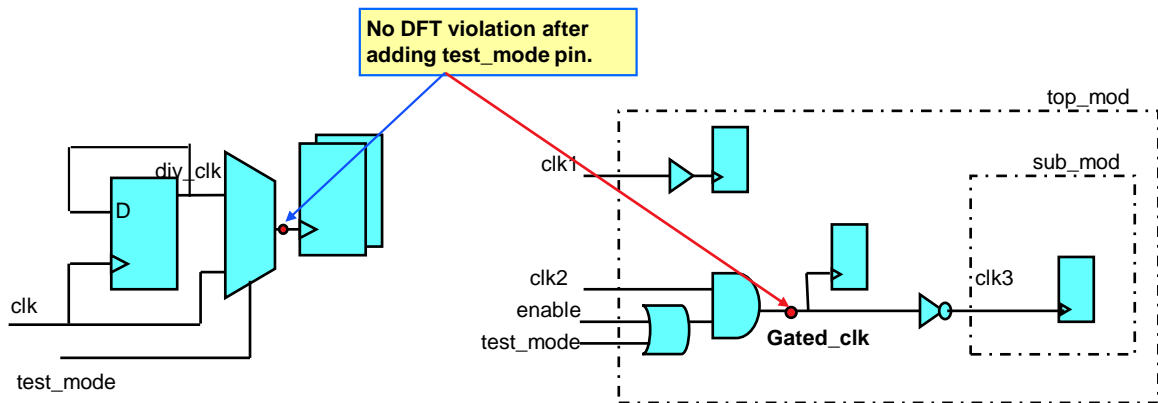
When you define a scan clock signal as ideal, the DFT engine sets the *ideal\_network* attribute to true on the pin or port for the scan clock signal. If you redefine the signal, this *ideal\_network* attribute is **not** reset automatically.

# 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 does not exist in the design.



04/25/11

Encounter RTL Compiler

217

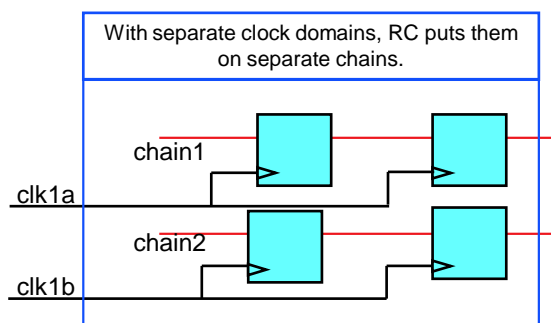
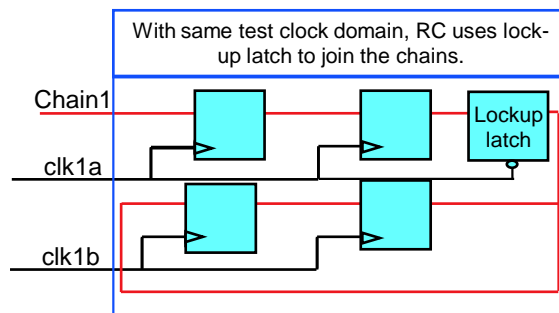
Use test-mode signals to bypass internally-generated clocks and enabled gated clocks and to inactivate asynchronous set or reset signals to pins of flip-flops during the scan-shift mode.

# 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.



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] /
```

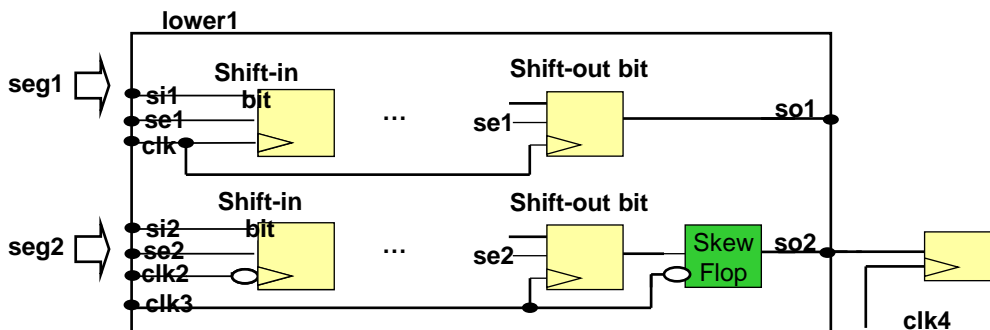
# Defining Scan Segments

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 {abstract_segment| fixed_segment
  | boundary_scan_segment | floating_segment
  | preserved_segment |shift_register_segment }
```

You need to define scan chain segments prior to running the DFT rule checker.

## Example of an Abstract Scan Segment



04/25/11

Encounter RTL Compiler

219

You can also use the `identify_shift_register_scan_segments` command to automatically identify shift register segments.

## Explanation of the Example of an Abstract Scan Segment

Command used:

```
define_dft abstract_segment -instance lower1 \  
  -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.
- ◆ [check\\_mbist\\_rules](#): Checks for MBIST rule violations.
- ◆ [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 Pattern 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%
```

04/25/11

Encounter RTL Compiler

221

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.

## Syntax

```
check_dft_rules [design] [-advanced]
[-max_print_violations integer | > file]
[-max_print_registers integer]
[-max_print_fanin integer]
[-dft_configuration_mode dft_config_mode_name]
```

# 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
lset_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 )
Reporting registers that are preserved or marked dont-scan
Reporting registers that are marked Abstract Segment Dont Scan
Reporting registers that are part of shift register segments
Reporting registers that are identified as lockup elements
Summary:
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 | -async_set | -async_reset  
  [-violations violation_object_id_list | -tristate_net | -xsource ]  
  [-preview] -test_control test_signal
```

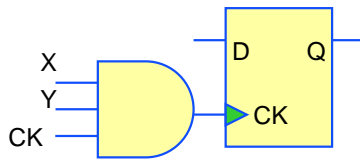
This command lets you do the following:

- ◆ Identify the type of violations to fix, either *-async\_set*, *-async\_reset*, *-clock*, *-tristate\_net*, or *-xsource*.
- ◆ The *-test\_control* option specifies the particular test signal to use to fix the violation. The added logic can now be controlled through this test signal.
- ◆ You can selectively fix any DFT violation by specifying its violation identifier (VID) using the *-violations* option,
- ◆ Display the modifications before making the fixes, through the *-preview* option.

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.

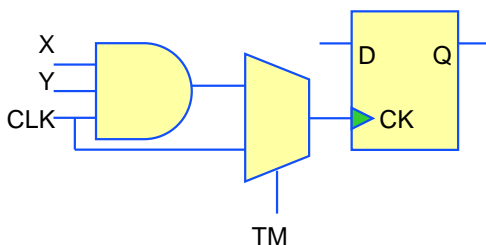


**Before: Clock is uncontrollable**

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**

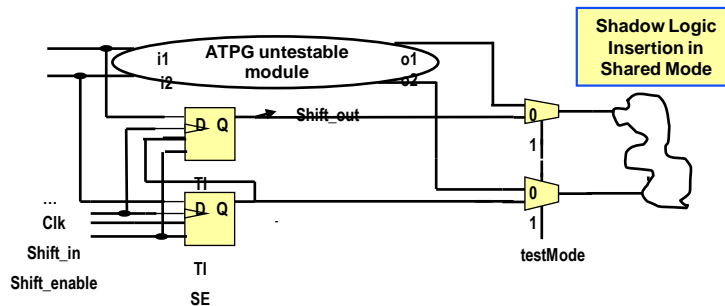
# Inserting DFT Logic

You can insert various DFT logic into your design using the *insert\_dft* command, mainly to improve the testability of your design.

## Syntax

```
insert_dft { analyzed_test_points | boundary_scan  
  | dfa_test_points | jtag_macro | lockup_element | mbist  
  | ptam | scan_power_gating | shadow_logic  
  | test_point | user_test_point | wrapper_cell }
```

## Example of adding shadow logic around



04/25/11

Encounter RTL Compiler

226

**boundary\_scan:** Inserts boundary scan cells and the corresponding JTAG controller.

**dfa\_test\_points:** Inserts test points based on Deterministic Fault Analysis.

**jtag\_macro:** Inserts a JTAG Macro controller into a netlist.

**lockup\_element:** Inserts lockup elements in the specified analyzed scan chains.

**mbist:** Inserts Memory Built-In-Self-Test (MBIST) logic to test targeted memories in the design.

**opcg:** Inserts On Product Clock Generation (OPCG) logic that generates on-chip launch and capture clocks for testing of at-speed delay defects.

**ptam:** Inserts Power Test Access Mechanism (PTAM) control logic into the design.

**rrfa\_test\_points:** Inserts test points based on Random Resistance Fault Analysis (RRFA)

**scan\_power\_gating:** Inserts gating logic at selected flop outputs to minimize switching power during scan shift

**shadow\_logic:** Inserts DFT shadow logic to enable testing of shadow logic around a module.

**test\_point:** Inserts a native test point.

**user\_test\_point:** Inserts a user-defined test point.

**wrapper\_cell:** Inserts an IEEE-1500 style core-wrapper cell.

# Preserving Nonscan Flops During Synthesis

---

- ◆ 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}  
/designs/design
```

- ◆ To preserve a complete subdesign, use:

```
set_attr dft_scan_map_mode preserve [find / -  
subdesign subname]
```

- ◆ Use the following attribute to prevent conversion of a flop to a scan flop.

```
set_attr dft_dont_scan true flip-flop_object
```

- ◆ Run the DFT rule checker to update the DFT status of all flipflops in the lower-level blocks attributed with *preserve*.

```
check_dft_rules
```

# Synthesize and Map to Scan

To synthesize the RTL design and map the flip-flops to their scan-equivalent flip-flops, type:

```
synthesize -to_mapped \  
  [-auto_identify_shift_register \  
  [-shift_register_min_length integer] \  
  [-shift_register_max_length integer]]
```

- ◆ 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.

## Scan mapping status report

```
=====
Scan mapping: converting flip-flops that pass TDRS.
Category Number Percentage
-----
Scan flip-flops mapped for DFT 11 100.00%
Flip-flops not mapped for DFT
non-DFT scan flip-flops 0 0.00%
flip-flops not scan replaceable 0 0.00%
flip-flops not targeted for DFT 0 0.00%
-----
Totals 11 100.00%
```

04/25/11

Encounter RTL Compiler

228

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}  
/designs/top_design
```



# Controlling Mapping to Scan in a Mapped Netlist

---

If you start with a structural netlist that was not mapped for scan, you can specify a one-to-one correspondence between the non-scan flop library cells and the scan flop library cells.

To run scan mapping on a mapped netlist, limit the changes made to the structural netlist, and map the nonscan flops to scan flops, use:

```
replace_scan [-to_non_scan] [-dont_check_dft_rules]  
[design]
```

- ◆ **-to\_non\_scan**: Replaces all scan flops that are part of shift register segments to non scan flops except for the first element in the segments.

To define a one-to-one correspondence between the non scan-flops and the scan flops use the `set_scan_equivalent` command.

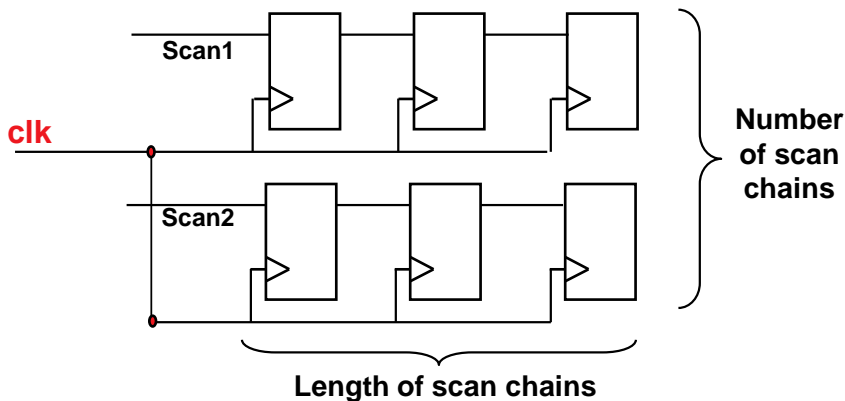
```
set_scan_equivalent -non_scan_cell libcell -scan_cell libcell  
[-tieoff_pins string] [-pin_map list_of_pin_groups]
```

Following initial synthesis, you can still fix the DFT rule violations in the mapped netlist by running the `fix_dft_violations` command.

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.

# Controlling Scan Configuration

- ◆ 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.



04/25/11

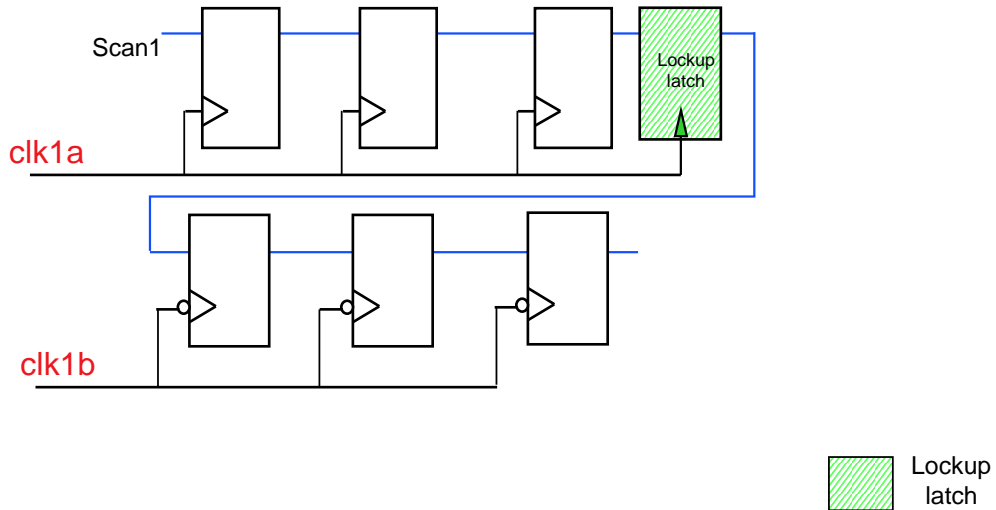
Encounter RTL Compiler

230

- 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`

## Controlling Scan Configuration (continued)

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.



04/25/11

Encounter RTL Compiler

231

- 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*.

# 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` has 16 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 32 /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`.

- ❑ 1 chain with 16 scan flip-flops of rising edge of test clock `clk1`
- ❑ 1 chain with 8 scan flip-flops of falling edge of test clock `clk1`
- ❑ 1 chain with 32 scan flip-flops of rising edge of test clock `clk2`

If the `dft_mix_clock_edges_in_scan_chains` attribute is set to `true`, then the configuration output is:

- 1 scan chain of 24 flops in test clock domain `clk1`, but with a lockup latch.
- 1 scan chain of 32 flops in test clock domain `clk2`

# 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]
```

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**

## Example Report

```
rc:/> report dft_chains
Reporting 5 scan_chains (clocked_lssd_scan)
Chain 1: AutoChain_1
scan_in: DFT_sdi_1
scan_out: DFT_sdo_1
length: 80
bit 1 out1_reg_0
bit 2 out1_reg_1
...
bit 80 out1_reg_79
-----
Chain 2: AutoChain_2
...
```

- ◆ To report the DFT setup, run the following command:

**report dft\_setup > [filename]**

# Generating Output Scan Files

---

- ◆ To create a scanDEF interface file for scan chain reordering in the Encounter™ system, use:

```
write_scandef > top_scan.def
```

- ◆ To create an ATPG interface file, use:

```
write_et_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.



# write\_template -split -dft -outfile run.tcl

What's new?  
dft\_run.tcl

```
#####
#   DFT SETUP (DFT design attributes &scan_chain, test clock definition ...)   #
#####
#####
## DFT Setup
#####

set_attr dft_scan_style muxed_scan/

## Uncomment for clocked_LSSD
#set_attributedft_scan_style clocked_lssd_scan/
#define_dftscan_clock_a-name <scanClockAObject> -period <delay in pico sec, default 50000> -rise <integer> -fall <integer> <portOrpin>
#define_dftscan_clock_b-name <scanClockAObject> -period <delay in pico sec, default 50000> -rise <integer> -fall <integer> <portOrpin>

set_attributedft_prefix DFT_ /
# For VDIO customers, it is recommended to set the value of the next two attributes to false.
set_attributedft_identify_top_level_test_clocks true /
set_attributedft_identify_test_signal true /

set_attributedft_identify_internal_test_clocks false /
set_attribute use_scan_seqs_for_non_dft false /

set_attributedft_scan_map_mode tdrp_pass "/designs/$DESIGN"
set_attributedft_connect_shift_enable_during_mapping ie_off "/designs/$DESIGN"
set_attributedft_connect_scan_data_pins_during_mapping oopback "/designs/$DESIGN"
set_attributedft_scan_output_preference auto "/designs/$DESIGN"
set_attributedft_lockup_element_type preferred_level_sensitive "/designs/$DESIGN"
#set_attributedft_mix_clock_edges_in_scan_chain true "/designs/$DESIGN"

#set_attributedft_dont_scan true <instance or subdesign>
#set_attributedft_controllable "<from pin> <inverting|non_inverting>" <to pin>

define_dfttest_clock-name <testClockObject> -domain <testClockDomain> -period <delay in pico sec, default 50000> -rise <int> -fall <int>
define_dftshift_enable-name <shiftEnableObject> -active <high|low> <portOrpin> [-create_port]
define_dfttest_mode-name <testModeObject> -active <high|low> <portOrpin> [-create_port] [-shared_in]
```

## dft\_run.tcl (continued)

```
## If you intend to insert compression logic, define your compression test signals or clocks here:
## define_dfttest_mode... [-shared_in]
## define_dfttest_clock...
#####
## Segments Constraints (support fixed, floating, preserved and abstract)
## only showing preserved, and abstract segments as these are most often used
#####
##define_dftpreserved_segment-name <segObject> -sdi <pin|port|subport> -sdo <pin|port|subport> -analyze
## If the block is complete from a DFT perspective, uncomment to prevent any nscan flops from being scanreplaced
##set_attr dft_dont_scantrue [filterdft_mappedfalse [find /designs/*-instance <subDesignInstance>/instances_seq*]]

##define_dftabstract_segment-name <segObject> <-module|-instance|-libcell> -sdi <pin> -sdo <pin> -clock_port<pin> [-rise|-fall] -shift_enable_port<pin> -
  active <high|low> -length <integer>
## Uncomment if abstract segments are modeled in CTL format
##read_dft_abstract_model-ctl <file>

define_dftscan_chain-name <ChainName> -sdi <topLevelSDIPort> -sdo <topLevelSDOPort> [-hookup_pin_sdi<coreSideSDIDrivingPin>] [-
  hookup_pin_sdo<coreSideSDOLoadPin>] [-shift_enable<ShiftEnableObject>] [-shared_output] [-non_shared_output] [-terminal_lockup<level | edge>]

## Run the DFT rule checks
check_dft_rules> $_REPORTS_PATH/${DESIGN}tdrcs
report_dft_registers> $_REPORTS_PATH/${DESIGN}DFTregs
report_dft_setup> $_REPORTS_PATH/${DESIGN}DFTsetup_tdr
## Fix the DFT Violations
## Uncomment to fixdft violations
## set numDFTviolations[check_dft_rules]
## if {$numDFTviolations> "0"} {
##   report_dft_violations> $_REPORTS_PATH/${DESIGN}DFTviols
##   fix_dft_violations-async_set-async_reset[-clock]-test_control<TestModeObject>
##   check_dft_rules
## }
```

## dft\_run.tcl (continued)

---

```
## Run the Advanced DFT rule checks to identify:
## ... x-source generators, internal tristate nets, and clock and data race violations
## Note: tristate nets are reported for busses in which the enables are driven by
## tristate devices. Use check_design to report other types of multidriven nets.

check_design-multidriven
check_dft_rules-advanced > $_REPORTS_PATH/${DESIGN}Advancedtdrcs
report_dft_violations[-tristate] [-xsource] [-xsource_by_instance] > $_REPORTS_PATH/${DESIGN}AdvancedDFTViols

## Fix the Advanced DFT Violations
## ... x-source violations are fixed by inserting registered shadow logic
## ... tristate net violations are fixed by selectively enabling and disabling the tristate enable signals
## in shift mode.
## ... clock and data race violations are not auto-fixed by the tool.
## Note: The fixing of tristate net violations using the fix_dft_violations-tristate_net command
## should be deferred until a full chip representation of the design is available.

## Uncomment to fix x-source violations (or alternatively, insert the shadow logic using the
## 'insert_dftshadow_logic' command).
#fix_dft_violations-xsource -test_control <TestModeObject> -test_clock_pin <ClockPinOrPort> [-exclude_xsource <instance>]
#check_dft_rules-advanced

## Update DFT status
## report_dft_registers > $_REPORTS_PATH/${DESIGN}DFTregs_tdr
## report_dft_setup > $_REPORTS_PATH/${DESIGN}DFTsetup_tdr
```

# run.tcl for write\_template –split –dft

```
##### Including DFT setup. (DFT design attributes & scan_chain, test clock definition etc..)
include dft_run.tcl
#####
## Optional DFT commands (section 1)
#####
#####
## Identify Functional Shift Registers
#####
#identify_shift_register_scan_segments
#####
## Add testability logic as required
#####
#insert_dftshadow_logic-around <instance>-mode <no_share|share|bypass> -test_control<TestModeObject>
#insert_dft_test_point-location <port|pin> -test_control <test_signal> -type <string>
#####
## Add Boundary Scan and MBIST logic
#####
## Uncomment to define the existing 3rd party TAP controller to be used as the master controller for
## DFT logic such as boundaryscan, compression, mbist and ptam.
#define_dftjtag_macro-name <objectName> ....
## Define JTAG Instructions for the existing Macro or when building the JTAG_Macro with user-defined instructions.
## ... For current release, name the mandatory JTAG instructions as: EXTEST, SAMPLE, PRELOAD, BYPASS
#define_dftjtag_instruction_register-name <string>-length <integer>-capture <string>
#define_dftjtag_instruction-name <string>-opcode <string> ;# [register <string>-length <integer>] [private]
## ... Uncomment if building a JTAG_Macro with MBIST instructions
## ... For current release, name the mandatory instructions as: RUN_MBIST, DIAGNOSE_MBIST, CONTINUE_MBIST
#define_dftjtag_instruction-name <string>-opcode <string> -register <string>-length <integer>
## Uncomment to define the MBIST clock if inserting MBIST logic
#define_dftmbist_clock-name <objectNameOfMBISTClock> ...
#insert_dftboundary_scan-tck <tckpin> -tdi <tdipin> -tms <tmspin> -trst <trstpin> -tdo <tdopin> -exclude_ports<list of ports excluded from
  boundary register>-preview
#insert_dftmbist-config_file<filename>-test_control<testModeObject> -shift_enable<shiftEnableObject> -connect_to_jtag-directory
  <MBISTworkDir> ..
## Uncomment to run the MBIST rule checks
#check_dft_rules-mbist-interface_file_dirs<InterfaceDirsForMBISTLogic>
## Write out BSDL file
#write_bsdl-bsdlout<BSDLfileName>-directory <work directory>
```

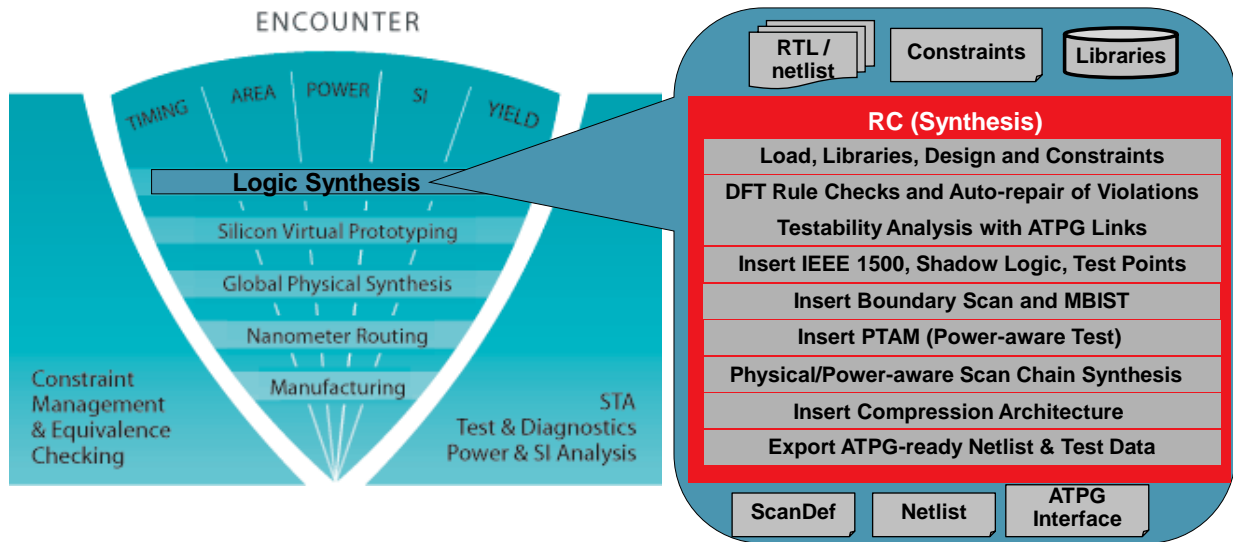
04/25/11

Encounter RTL Compiler

240

```
#####
## Optional additional DFT commands. (section 2)
#####
## Re-run DFT rule checks
## check_dft_rules[-advanced]
## Build the full scanchains
## connect_scan_chains[-preview] [-auto_create_chains]
## report_dft_chains> $_REPORTS_PATH/${DESIGN}DFTchains
## Inserting Compression logic
## compress_scan_chains-ratio <integer> -mask <string> [-auto_create] [-preview]
##report_dft_chains> $_REPORTS_PATH/${DESIGN}DFTchains_compression
#####
## DFT Reports
#####
report_dft_setup> $_REPORTS_PATH/${DESIGN}DFTsetup_final
write_scandef> ${DESIGN}-scanDEF
write_atpg[-stillmentor|cadence] > ${DESIGN}-ATPG
write_dft_abstract_model> ${DESIGN}-scanAbstract
write_hdl-abstract > ${DESIGN}LogicAbstract
write_script-analyze_all_scan_chains> ${DESIGN}-writeScript-analyzeAllScanChains
## check_atpg_rules-library <Verilog simulation library files>compression-directory <Encounter Testworkdir directory>
## write_et_bsv-library <Verilog structural library files>directory $SET_WORKDIR
## write_et_mbist-library <Verilog structural library files>directory $SET_WORKDIR-bsv -mbist_interface_file_dir<string> -
  mbist_interface_file_list<string>
## write_et_atpg-library <Verilog structural library files>compression-directory $SET_WORKDIR
```

# Advanced DFT Features



04/25/11

Encounter RTL Compiler

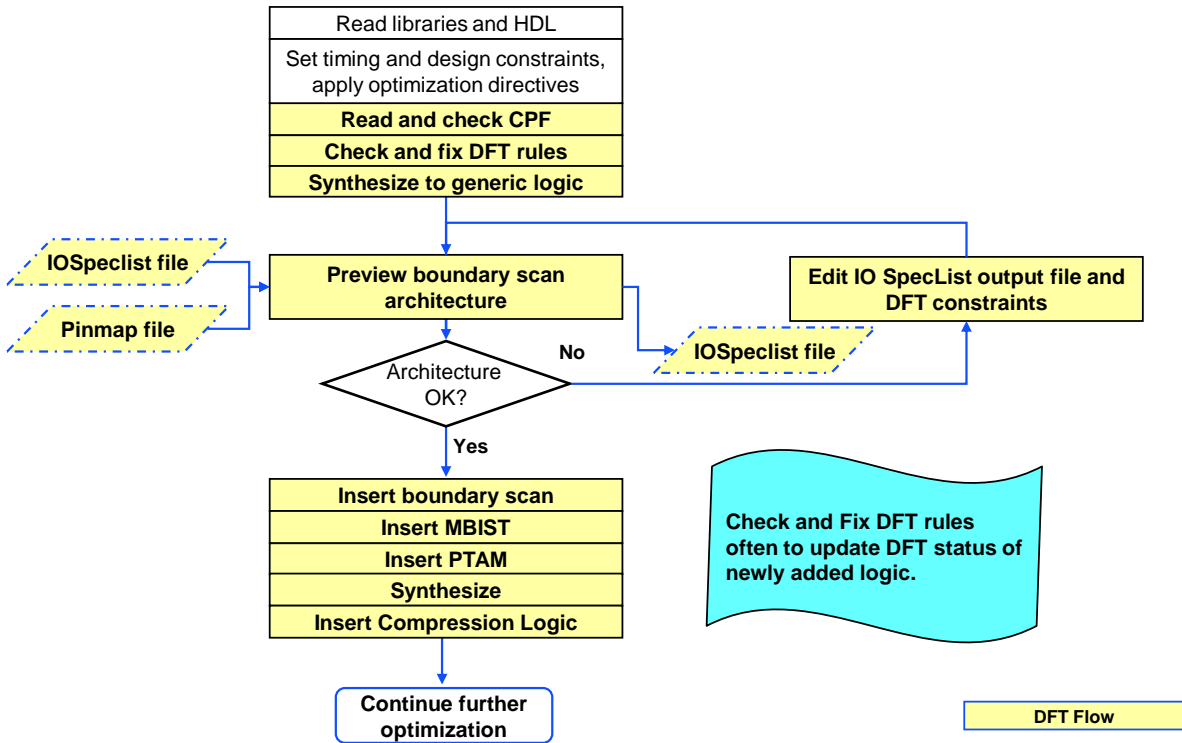
242

RTL Compiler is an integral component of our Encounter Platform Solution.

Here we see an overview of the Advanced DFT features now available natively in RTL Compiler:

The advantages of combining leading RTL synthesis technology with the industries most advanced DFT capabilities provide the highest quality netlist, testability, and predictability. The details are discussed in the following foils.

# Advanced Design For Test Synthesis



04/25/11

Encounter RTL Compiler

243

The *power test access mechanism* (PTAM) logic stabilizes the power test mode for test. Insertion of overriding control logic into the power manager and enables test application control over selected power management enable pins described in the *common power format* (CPF) file.

As the size of integrated circuits grow, traditional full scan ATPG vectors take up a lot of *automatic test equipment* (ATE) memory and test time to apply the vectors to the *device under test* (DUT). In addition, newer chips often have more pins and functionality that cannot be handled by older ATE equipment that have both a limited number of pins and limited buffer memory.

- To reduce the ATE runtime, you can increase the number of scan chains in the design. However, without any other changes to the design, this results in a need for more test pins and the ATE might not have enough test pins available.
- To reduce the test data volume, you can reduce the number of patterns but that would negatively affect the test coverage.
- By using *test compression*, you can reduce the ATE test times and test data volume without compromising test coverage of the design. Inserting compression logic involves adding a decompressor and compressor.
- For test input data decompression, the tool can insert broadcast-scan and an optional XOR-based test input spreader.
- For the test output data compression, the tool can insert either an XOR-based compressor or a MISR-based compressor.

# Full-Chip DFT Integration

Concurrent one-pass synthesis

Structural verification (2-Layer)

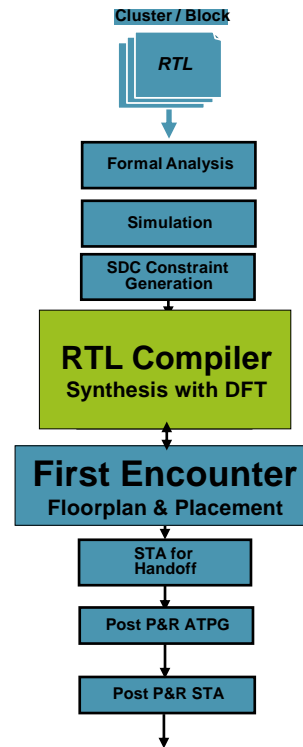
- ◆ At both RTL and Gate Level
- ◆ DFT rule check and auto-repair

Physically-aware DFT

- ◆ Scan inserted during mapping
- ◆ Placement-aware scan order
- ◆ DFT-aware clock gating and insertion
- ◆ Compression architecture inserted

Power-aware DFT

- ◆ MV & MTCMOS support
- ◆ Test for level shifters & isolation logic
- ◆ Power Test Access Module (PTAM)
- ◆ Scan insertion, clock-gating





# Full-Chip DFT Integration (continued)

Link to Encounter True Time (ATPG)

Flexible DFT compression insertion

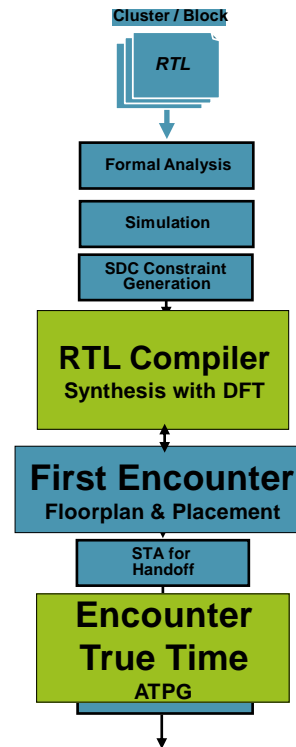
Robust channel masking capability in XOR and MISR compression

Enhanced test coverage and analysis

- ◆ Efficient test point insertion (DFA)
- ◆ Shadow logic insertion
- ◆ Testability analysis

ATPG-ready netlist and scripts

- ◆ Build netlist model and fault model
- ◆ Build test mode
- ◆ Create test patterns
- ◆ Generate ATPG test bench
- ◆ Validate test patterns using Simulation



# RC DFT Rule Check, Debug, and Auto-Fix

## GUI Interface Within RC Provides Ease of Use

### DFT Rule Checker

- ◆ Ultra fast checker (100K flops in a few seconds)
- ◆ Handles test mode setup signals
- ◆ Analyzes through clock-gating logic and latches
- ◆ Identifies test clock domains
- ◆ Links to ATPG rule checker
- ◆ Feedback on DFT violations at RTL/gates/schematic

### GUI Interface for DFT Debug

### Auto-Fix DFT Violations

**Schematic view of logic causing the violation**

**Link back to RTL code that causes violation**

**Table showing all DFT Violations**

ID	Type	Description	Module	Net	Pin
3	clock	clock signal driven to a constant value	bot	bck	bout_reg/clk
4	clock	clock signal has no driver	mid	fck	b6v/bck
5	clock	detected a loop while tracing clock signal	mid	lck	g15/z
6	clock	internal or gated clock signal	top	gck	g1/z
7	async	async signal driven by a sequential element	bot	bout	bout_reg/q
8	async	async signal driven to a constant active value, possibly due to a polarity conflict	mid	pre1	b6v/async
9	async	async signal has no driver	mid	mpre	b7z/async
10	async	async signal driven to a constant active value, possibly due to a polarity conflict	mid	ck	top/ck
11	async	async signal also used as a clock signal	top	ck	top/ck
12	async	async signal driven to a constant active value, possibly due to a polarity conflict	mid	mpre	b6v/async
13	async	detected a loop while tracing async signal	mid	lpre	g14/z

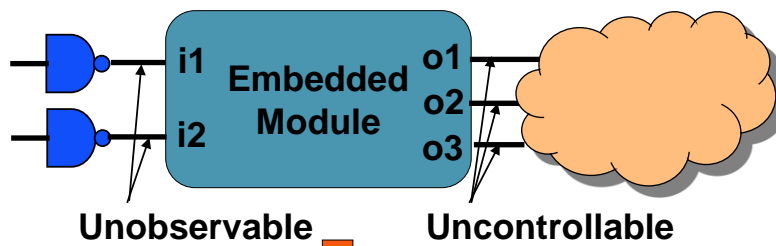
04/25/11

Encounter RTL Compiler

246

We offer an Ultra-fast DFT rule checker that identifies all critical DFT violations. GUI provides a mechanism to link back the source of DFT violations to the original RTL and to the netlist schematic for easy debug. It also allows user to automatically fix the identified DFT violations.

# Shadow Logic and Test Point Insertion



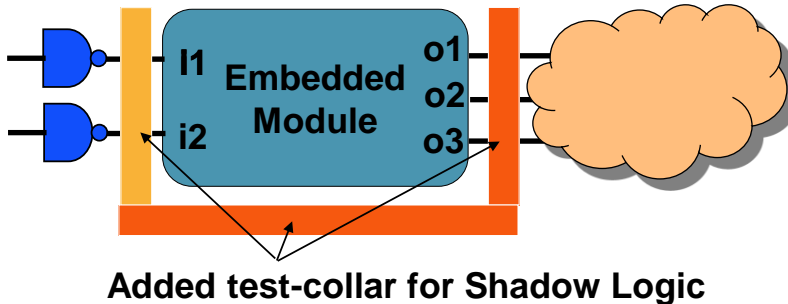
## Challenge:

- Embedded Modules block testability of Shadow Logic (surrounding logic)



## Solutions:

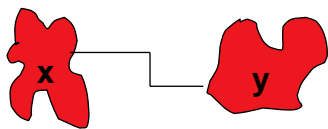
- Insert control and observation test points
- Scannable elements or Bypass logic



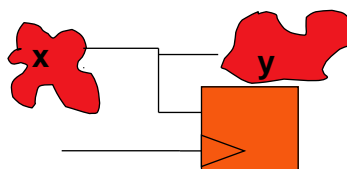
Embedded modules can make it harder for ATPG to generate tests for the logic surrounding the modules – called “Shadow Logic”. RC can insert different test collars (bypass logic, or control and observation test points) around such blocks to improve testability of the Shadow Logic.

# Boosting Fault Coverage and Reducing Test Patterns (Deterministic Fault Analysis)

## Circuit *before* test point insertion

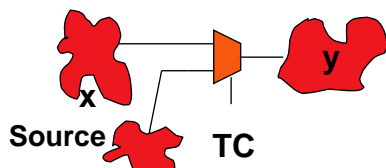


## Scannable observation test point

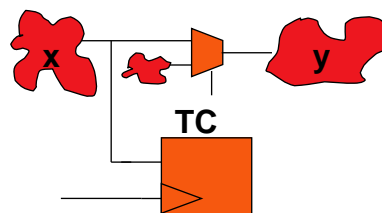


**15 Different test point types supported**  
**Also supports any arbitrary user-defined test point structure**

## Control test point



## Scannable observation and control test-point



04/25/11

Encounter RTL Compiler

248

Here we show different examples of test points that RC can insert. Users can choose from 15 different pre-defined test point structures, or define their own custom test point structure.

# Block-level Embedded Core Testing

## Improve testability with leading IEEE 1500 solution

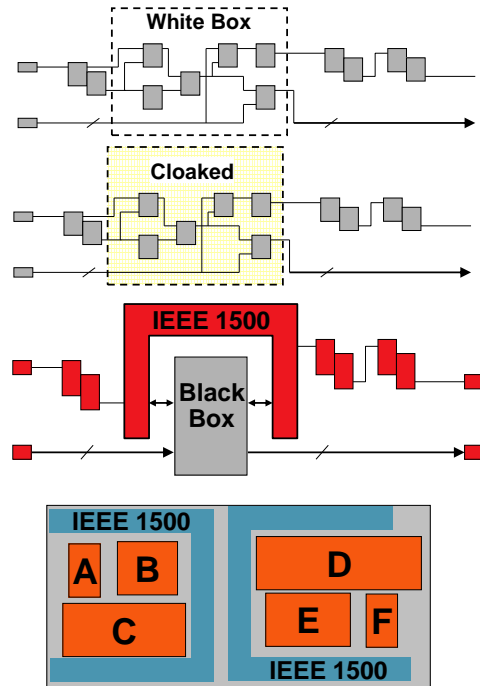
### Multiple implementation modes

- ◆ White box – totally visible
- ◆ Cloaked – hidden from user, not tool
- ◆ Black box wrapper – totally hidden

Insertion of dedicated and shared wrappers

Improves full-chip test coverage

Allows partitioning for DFT & ATPG



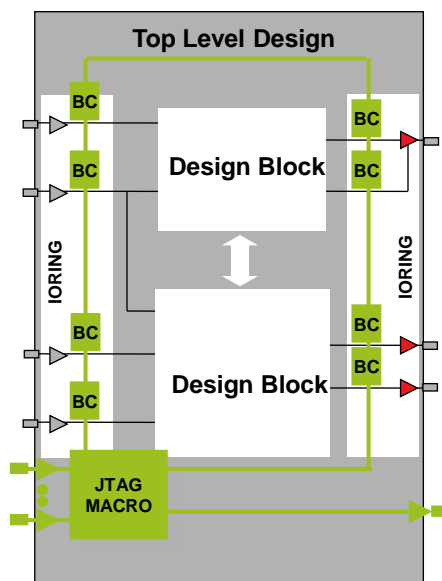
To improve test-mode access to embedded cores, RC can insert IEEE-1500 style core-wrapper cells around such blocks. It supports both – shared and dedicated wrapper cells. This feature can also be used to partition a big design for DFT purposes. This allows generation of smaller partitions that can have independent scan chains and ATPG flow.

# Top-level Boundary Scan Integration

Automated IEEE1149.x insertion:

- ◆ TAP state machine and ports
- ◆ Bypass, IDCODE, USERCODE registers and Custom Instructions
- ◆ Integration with MBIST and PTAM
- ◆ Supports Embedded I/O pads
- ◆ Generates RTL or mapped netlist for IEEE 1149.1 TAP controller

Full verification and BSDL generation



This slide summarizes the capabilities for IEEE 1149.x boundary scan.

Scan insertion capability is based on Encounter RTL compiler technology. Encounter Test Architect takes care of any required hierarchical connections to the TAP controller at the top level.

Note that a test product (TDE001 or ET001) is required when the function is invoked from the RTL compiler environment

# Physically-aware Scan Chain Optimization

## Challenges:

- Lack of physical information
- Scan chain congestion
- Impact on timing/SI

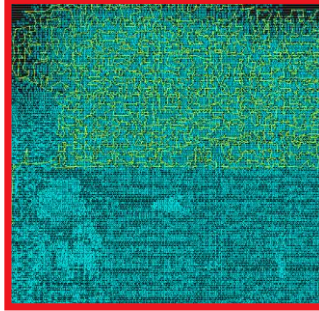
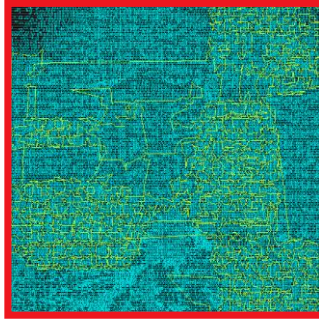


## Solution:

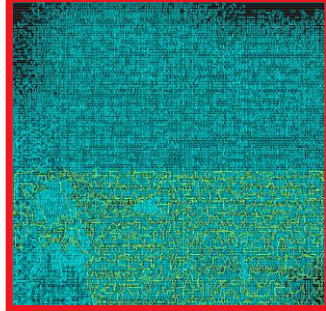
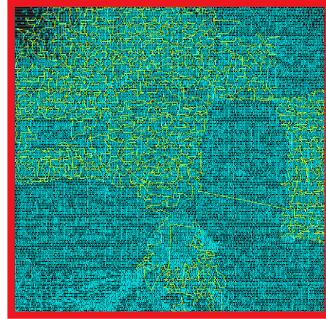
- Scan chain build with physical information from First Encounter or SoC Encounter
- Proven to reduce scan wire congestion by 40%
- Improved balancing
- Scandef available

```
read_def design.def
connect_scan_chains -physical
```

Scan Chain A



Scan Chain B



Physically-aware DFT allows RC to insert much better scan chains that are localized to an area over the chip – thus, reducing the scan wire and associated congestion.

RC also generates scandef for standalone physical reordering of scan chains (so tools are very flexible and can work within customized flows with impressive results)

# Power Aware DFT for Testing Power Modes

## “Ad Hoc” LP Test Challenges:

### Low productivity

- ◆ No ATPG based flow for LP designs
- ◆ Relies on functional vectors

### Low Test Coverage

- ◆ Power Features “overridden”
- ◆ Power structures not targeted (LS,SR)

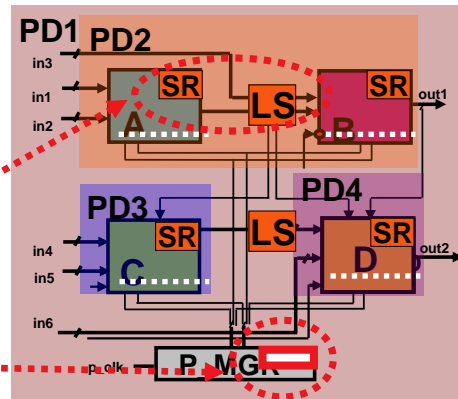
## Solution:

### Power-Aware DFT understands

- ◆ Power modes, domains, structures
- ◆ Power domain control under test with insertion of Power Test Access Mechanism (PTAM)

### Power-Aware ATPG

- ◆ Test power modes, domains (pwr dwn)
- ◆ Verify isolation target test structures
- ◆ Minimize power during test



04/25/11

Encounter RTL Compiler

252

The before Power Ware Test situation is as follows:

Current Test Solutions do not support low power design:

- Power Features “overridden”
- Testing with all Power Modes “on”
- Power structures not specifically targeted
- Power shut off
- Isolation
- State retention (SR)

## “Ad Hoc” Test Strategy

### Low productivity:

- No ATPG based flow for testing low power designs
- Relies on functional vectors to test power functions

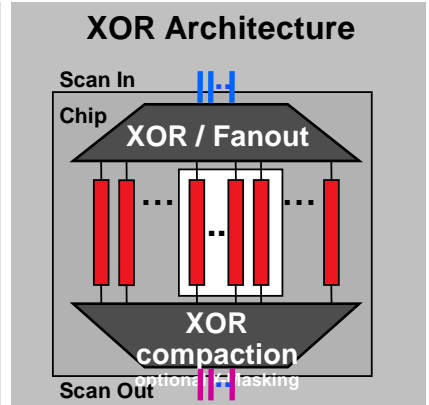
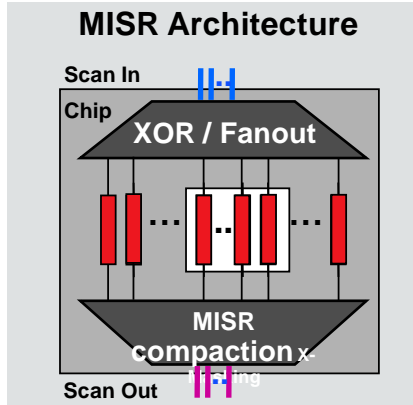
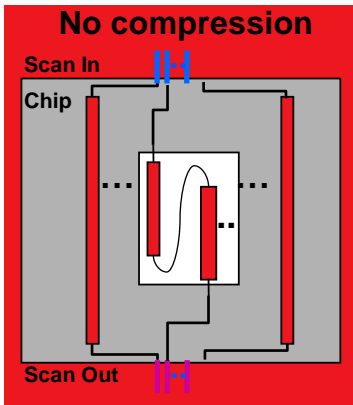
### Low Test Coverage

- Power Features “overridden”
  - Testing with all Power Modes “on”
  - No test for Isolation circuits
- Power structures not specifically targeted
  - Level shifters (LS)
  - State retention (SR)



# Flexible RTL Compiler Compression Strategies

## Advanced Masking Algorithms for High Pattern Efficiency



Data Vol Reduction vs no comp	Typically up to 150X	Typically up to 50X
Test Time Reduction vs. no comp	Typically up to 100X	Typically up to 50X
Diagnostics methodology	Two pass (off-line) for full data	One pass (on-line)
Diagnostics accuracy	Best with full data	Effective W/O full data

# MBIST Insertion and Execution

## Flexible memory test solution

### Challenge:

- Sharing capabilities from 3<sup>rd</sup> Party
- Lack of comprehensive fault testing

### Solution:

- Robust fault coverage
- Easy BIST engine sharing
- Automatic insertion & stitching
- Failure analysis & redundancy support
- Leading embedded memory support
- Integrated with 1149.1 TAP

### Support for

- ◆ 1-port (1RW)
- ◆ 2-port (1R/1W and 2RW)
- ◆ ROM

Algorithm	Faults							
	Stuck-at	Transition	Address decoder	Stuck-op	Neighborhood Coupling	Bridging	Data retention	Recovery
March test	✓	✓	✓	✓	✓	✓	✓	✓
Checkerboard	✓	✓			✓	✓	✓	✓
Word line stripe	✓	✓			✓			✓
Galloping ones	✓	✓	✓	✓	✓	✓		
Pseudo-random address	✓	✓			✓	✓	✓	
Port interaction	✓	✓	✓		✓	✓		

Supported algorithms and fault models

04/25/11

Encounter RTL Compiler

254

RC includes a memory BIST compiler with robust fault coverage, easy BIST engine sharing, automatic insertion into netlists, and failure analysis support. The major features of this capability are:

Support for industry standard embedded memory architectures from leading suppliers, including full validation with ARM (Artisan) memory compilers

Support includes one-port and two-port SRAMs, register files and ROM

Control via IEEE 1149.1 TAP interface to enable a common test suite and access method from design verification through product manufacturing test through field testing.

Memory description in industry standard .lib format file

Support BIST macro sharing via binding of target memories to BIST engines.

Algorithm design complements available detailed physical neighbor knowledge and cell structure.

BIST engine generation supports test set reductions when such knowledge is available to the test application, reducing overall test time.

Design verification testbenches for use at the chip level; formal verification of pre-DFT versus post-DFT insertion netlists; logical failure analysis bit mapping.

Failure analysis is supported using error information retrieved from the Diagnostic Test Data Register within the BIST engines.

Memory redundancy analysis capabilities support row and/or column reconfigurable memory devices.

# define\_dft subcommands in RC

---

## define\_dft <subcommand>:

<b>abstract_segment</b>	- defines an abstract scan segment
boundary_scan_segment	- defines a boundary-scan segment
dft_configuration_mode	- define a mode for DFT configuration purposes
domain_macro_parameters	- defines the parameters of anopcg domain macro
<b>fixed_segment</b>	- defines a fixed-ordered scan segment
<b>floating_segment</b>	- defines a reorderable scan segment
jtag_instruction	- defines a jtag instruction
jtag_instruction_register	- defines a jtag instruction register
jtag_macro	- defines a jtag macro
mbist_clock	- defines a mbist clock
opcg_domain	- defines an opcg domain
opcg_mode	- defines an opcg mode for Encounter Test ATPG
opcg_trigger	- defines an opcg trigger source
osc_source	- defines an opcg oscillator source
<b>preserved_segment</b>	- defines a pre-connected scan segment
<b>scan_chain</b>	- defines a scan chain
scan_clock_a	- defines scan_clock_a for LSSD scan style
scan_clock_b	- defines scan_clock_b for LSSD scan style
<b>shift_enable</b>	- defines a shift enable signal
shift_register_segment	- defines a functional shiftregister as a scan segment
<b>test_clock</b>	- defines a test clock
<b>test_mode</b>	- defines a test mode signal

# DFT Feature/Command Listing

Feature	RC Command	ET License required
Inserting Boundary Scan	insert_dft boundary_scan	ET Architect Basic
Inserting MBIST	insert_dft mbist	ET Architect Advanced
Inserting PTAM	insert_dft ptam	ET Architect Advanced
Inserting Shadow Logic	insert_dft [shadow_logic   analyzed_test_points-shadow_logic]	None
Inserting 1500 cells	insert_dft wrapper_cell	None
Inserting FULLSCAN chains	<b>connect_scan_chains</b>	None
Inserting COMPRESSION	compress_scan_chains	ET Architect Advanced
Inserting OPCG	insert_dft opcg	ET Architect Advanced
Inserting Test Points	insert_dft [test_point   user_test_point   analyzed_test_points]	None ET Architect Basic
Inserting RRFA Test Points	insert_dft rrfa_test_points	None
Inserting Deterministic Test Points	insert_dft dfa_test_points	None

## DFT Feature/Command Listing<sub>(continued)</sub>

Feature	RC Command	ET License required
Checking DFT rules	check_dft_rules	None
Checking DFT rules advanced	check_dft_rules-advanced	ET Architect Advanced
Checking ATPG rules	check_atpg_rules	None
Fixing DFT violations	fix_dft_violations	None
Analyzing Testability	analyze_testability	True-Time Basic
Analyzing Scan Compressibility	analyze_scan_compressibility	True-Time Advanced
Identify Fixed-Value Registers	identify_test_mode_registers	None
Scan Power Estimation	report_scan_power	None
Interface files to ET	write_et_atpg, write_et_bsv, write_et_dfa, write_et_mbist, write_et_rrfa	ET True-Time Basic
Interface files to Encounter	write_scandef	None

04/25/11

Encounter RTL Compiler

257

# Lab Exercises

---



## Lab 9-1 Running Scan Synthesis

- Setting up for DFT rule checker
- Running DFT rule checks
- Running Scan Synthesis
- Connecting Scan Chains

# Completing the Post Class Assessment

---

1. In a web browser enter: <http://exam.cadence.com>
2. Login to the exam server:
  - a) **Name:** your complete email address (example: [joe@cadence.com](mailto:joe@cadence.com))
  - b) **Group:** your company's email suffix (example: [cadence.com](mailto:cadence.com))
3. Select the assessment with the title of:  
ES <your course title> **POST**
4. Complete the assessment.
5. Click Submit at the bottom of the exam. *Note: You will be given a score and the correct answers. We will discuss these following the exam.*

# How to Obtain a Certificate of Completion

## Instructor-Led or Virtual Course

1. Log in to <http://learning.cadence.com>, using your user name and password.  
If you have problems logging in, email [training\\_enroll@cadence.com](mailto:training_enroll@cadence.com).
2. Choose **Catalog – Advanced Catalog Search** from the pull-down menus. Enter the course title in the Title field and click **Search**.
3. Locate the item with *Exam* in the title.
4. Click **Request Approval**. On the new page that appears, click **Submit**.
5. After receiving approval via email from Cadence Training, again search for the course as above and then click **Go to Content**. The exam launches.
6. Complete the exam. After completion, your score will be displayed. A passing score is 75%.
7. If you passed, download the Certificate of Completion. Under the **Learning** tab, click **Learning History**, find the course, and click **Print Completion Certificate**.  
If you did not pass, you have the option of taking the exam again.

## iLS Course

1. Log in to <http://learning.cadence.com>, using your user name and password.  
If you have problems logging in, email [training\\_enroll@cadence.com](mailto:training_enroll@cadence.com).
2. Find the course in your learning plan. (If necessary, choose **Learning – Learning Plan** from the pull-down menus.)
3. Click **Go to Content**.
4. Locate the item with *Exam* in the title. Click the link to launch the exam.
5. Complete the exam. After completion, your score will be displayed. A passing score is 75%.
6. If you passed, download the Certificate of Completion. Under the **Learning** tab, click **Learning History**, find the course, and click **Print Completion Certificate**.  
If you did not pass, you have the option of taking the exam again.

04/25/11

Encounter RTL Compiler



# Thank You!

Log In | Register | Resource

Cadence Design Systems - Windows Internet Explorer

http://www.cadence.com/us/pages/default.aspx

File Edit View Favorites Tools Help

Favorites

Cadence Design Systems

cadence®

Solutions Products Services

Log I

EDA360

ARM and Cadence Colla

Joint effort will streamline development time of ARM processor-based

LEARN MORE ▶

Cadence Online Support

Software Downloads

Computing Platform Support

University Software Program

Training


Training Offerings

Training Course Catalogs

Support & Training Home

04/25/11

Encounter RTL Compiler



---

Blank Page

04/25/11

Encounter RTL Compiler

# Physical Synthesis

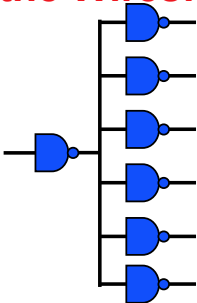
## Appendix A



April 25, 2011

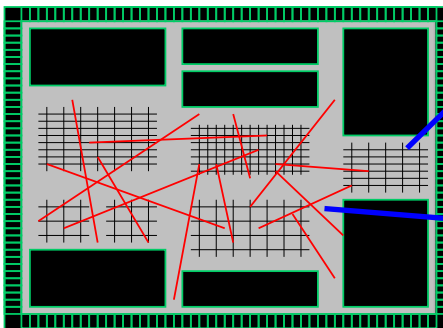
# Why Can't Some RC Results Close Timing in the Back End?

## It's the Wires!



### Synthesis view

All wires of fanout= $n$  are the same (for a given block size)



80-90% of wires are local

10% are big problems!

### Physical view

Each wire is unique

Synthesis tools optimize for delay, without accurate wire delay info.

Common workaround - add timing margin to synthesis

- ◆ Typical margin today: 30% global over-constrain
- ◆ Over-powers 80-90% of the design
- ◆ No longer practical in today's technologies

04/25/11

Encounter RTL Compiler

264

The root of the problem is wires. At 130nm and below, wires dominate the delay equation, and it gets worse with each process generation. Synthesis tools are tasked with creating an optimal logic structure, but they still rely on fanout-based wireload models that treat all wires for a given block size as the same. The reality is that every wire is unique. In a typical chip, 80-90% of wires are local interconnect, but even those can have different characteristics as you see here, depending on the nature of the block (shape, size, density, datapath, random logic, etc). The 10% wires that represent global interconnect are the real timing headaches.

So wireload models have become so far off the mark that many designers just ignore wire effects altogether – not a better solution, just a faster path through non-convergent iterations. The real challenge is to give synthesis tools better wire information...however how do you model wires before you have gates?

# The Need for Increasingly Accurate Interconnect Modeling

---



04/25/11

Encounter RTL Compiler

265

# Data File Types for RC Physical

## Wireload Model :

- ◆ liberty .lib files with WLM's

## RC-PLE & RC-Spatial:

- ◆ Mandatory:
  - ❑ Tech\_lef and standard cell .lef files
  - ❑ Liberty .lib standard cell files
- ◆ Optional, but recommended:
  - ❑ Captable .captable file
  - ❑ Floorplan .def file

## RC-Physical:

- ◆ Mandatory:
  - ❑ Tech\_lef and standard cell .lef files
  - ❑ Liberty .lib standard cell files
  - ❑ Floorplan .def file
- ◆ Optional, but recommended:
  - ❑ Captable .captable file

# Physical Layout Estimation (PLE)

What is PLE?

- ◆ A physical modeling technique to capture timing closure P&R tool behavior for RTL synthesis optimization

- **Result: better timing-power-area balance**

Uses actual design and physical library info

Dynamically adapts during optimization to changes in logic structures

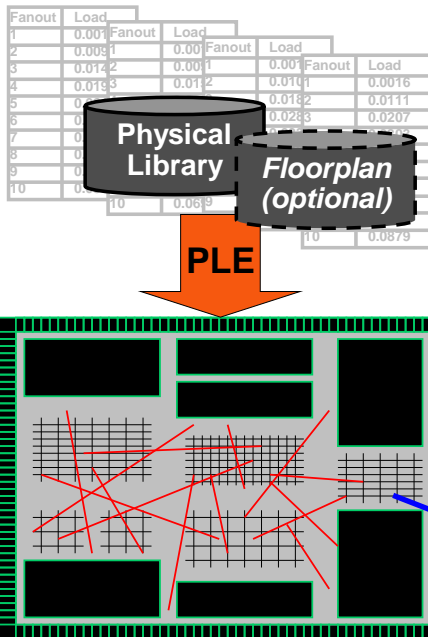
Same runtime as WLM

```
set_attribute lef_library <lef file(s)>
```

```
set_attribute cap_table_file <cap table>
```

**Does a good job modeling the short wires in a design (80-90%)**

***Improves QoS and predictability over WLM***



The process of going from RTL to gates has proven to make a big difference in the ultimate predictability of physical implementation. But how do you predict interconnect delay if you do not yet have gates to place?

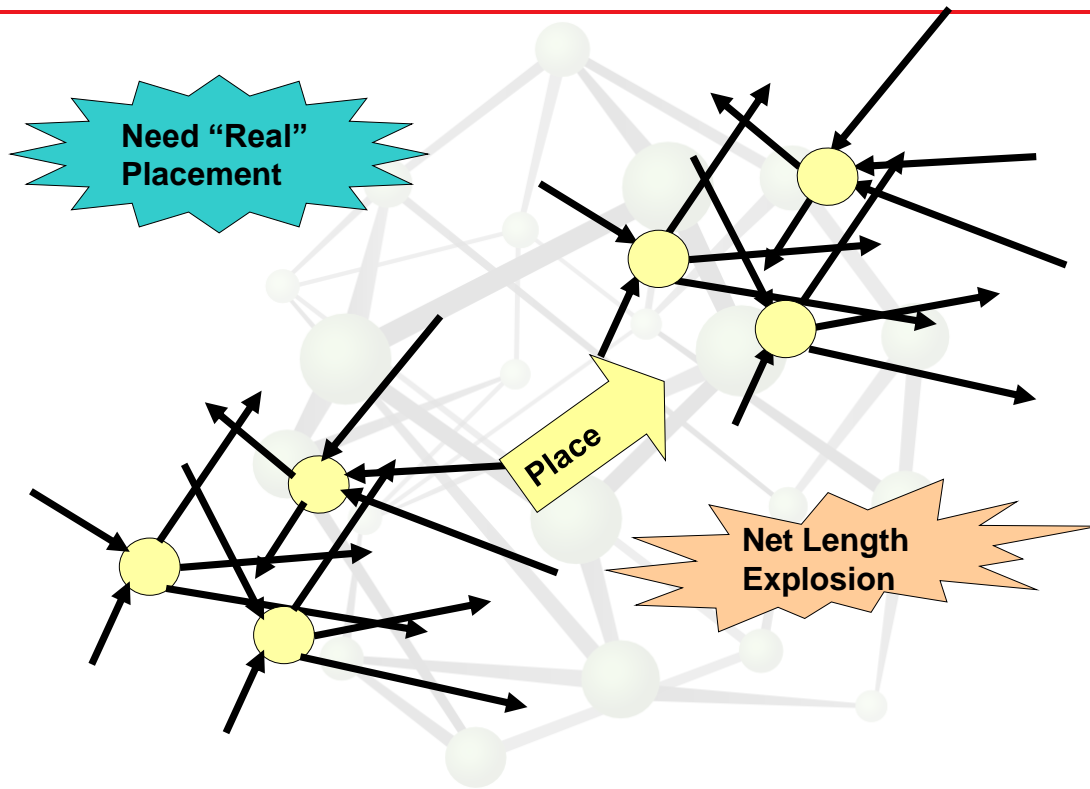
This is why wireload models have survived for so long – it is a difficult task. Customers wanting to get the best results still spend a lot of time today experimenting and trying to figure out which wireload model will get the best results out of physical design for each block.

We have devised a method called “Physical Layout Estimation” that is a physical modeling technique that can be used during the RTL structuring process. It uses actual physical library info (LEF) and capacitance tables that provide length-based capacitance info, and it adjusts based on changing design sizes and structures, throughout the optimization process. Because it can dynamically adapt, we find it provides the best results. Because it is pre-placement, it is not aware of what the long wires will be, however it does a good job modeling the local interconnect, no matter what the characteristics of the logic are.

It is easy to use – you just load the physical library information instead of wireload models, and the runtimes are the same as WLM synthesis too.

And we have recently added the option to read in a floorplan, to add some coarse-grained placement knowledge, like aspect ratio, pin location, macro location. Again, we’re not doing placement, but we can adjust the model if we know a path goes to a RAM or an I/O.

# RC-Spatial: Wasn't PLE Good Enough?!



04/25/11

Encounter RTL Compiler

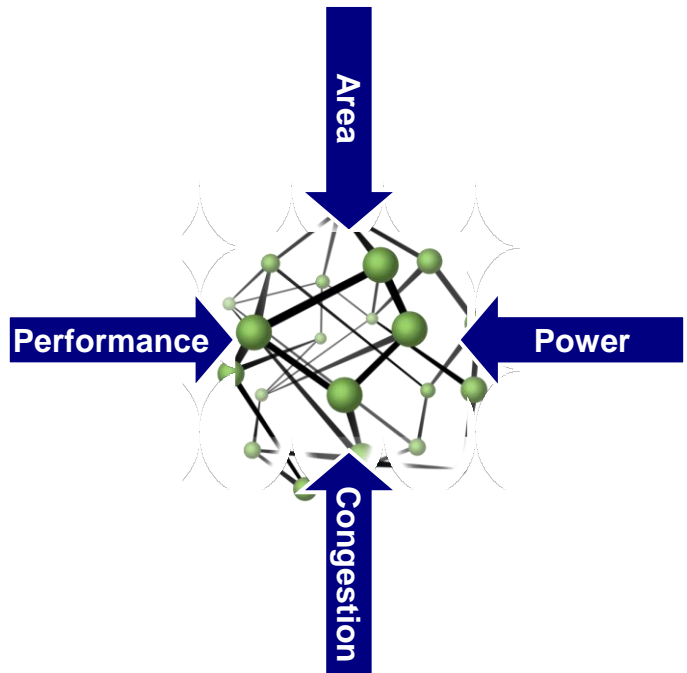
268

So this begs the question, why, or more accurately when, is PLE not good enough. To answer that we have to understand that there are different levels of requirements from different customers. Some want to model the wires only accurately enough so that the synthesis tools make good structuring decisions. They don't really care about how close that timing is to the physical implementation, they just want to get a "reasonable" netlist to take into their back-end flow. For these customers PLE typically is good enough, especially if the floorplan is close to square and there are not a ton of macros. The second set of customers want more predictability, but either don't want to or cannot hand off a DEF to the physical design team. They want "good" correlation, but understand that the ultimate correlation will depend on the production placement. This is when PLE is not good enough because PLE's are calculating individual nets and making assumptions about the placement. What typically happens in a real design is that the placer cannot achieve this assumptions about an "optimal" placement and the gates move apart, subsequently making the nets longer that could be predicted with PLE's alone. By adding quick placement to RC-Spatial, RTL-Compiler can get closer to predicting the actual preCTS timing. Of course the ultimate solution is to use RC-Physical, run legal, production placement and optimization in RC and then hand off that placement to the physical design team, but we will talk about that later.



# Spatial Technology

- ◆ This technology adds fast coarse placement to the physical layout estimation (PLE) model.
  - ❑ It provides better modeling of long wires.
  - ❑ Congestion based on net weighting is used as a global cost function.
- ◆ With spatial mode, the runtime is much faster than Encounter RTL Compiler with physical.
- ◆ It works on any license.



04/25/11

Encounter RTL Compiler

269

Spatial technology is the next generation of wire modeling using PLE. The spatial mode uses a fast, coarse-grained placement to identify where the long wires are. It is beneficial to provide a floorplan to locate the pins and macros.

This technology adds even more physical reality to RC's global cost functions, providing more directed optimizations, and giving more confidence to the synthesis user.

# Running RC-Spatial

---

Perfect for block designers

- ◆ Good timing prediction
- ◆ Quick “physical” feedback
- ◆ Estimate on-chip “space” requirements
- ◆ Little physical background required

Will work with or without floorplan

- ◆ Although a floorplan is recommended

## Syntax

```
synthesize -to_mapped -spatial -effort [low|medium|high]
```

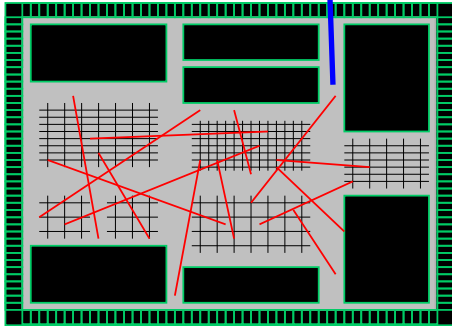
- ◆ Same as:

```
synthesize -to_mapped -effort [low|medium|high]
```

```
synthesize -to_mapped -spatial -effort [low|medium|high]
```

# Long Wires Solution: synthesize –to\_placed

- ❑ Long wires are dependent on the physical implementation of:
  - ❑ Floorplan, Placement &
  - ❑ Routing (congestion)
- ❑ Requires Silicon Virtual Prototype



## Encounter RTL Compiler with Physical

Synthesis w/ DFT w/ PLE

synthesize –to\_placed

Analyze / re-optimize

## First Encounter SVP



Full-chip virtual prototype

- Use production floorplan
- Tapeout-quality placement
- Trial route

Single command in RTL Compiler runs First Encounter full-chip silicon virtual prototyping (SVP).

Brings back relevant information into synthesis environment.

- ◆ Analyze physical timing in synthesis
- ◆ Incremental optimization w/ physical timing
- ◆ Can hand off placement for deterministic closure

04/25/11

Encounter RTL Compiler

271

Really the only way to identify the long wires on a chip is to use the production floorplan and production placement, with the actual DFT logic inserted as well, since high-fanout/fanin structures like compression and JTAG will impact physical timing. Trying to predict any other way will fail because these wires are dependent on those two factors.

However it is a physical design use model and cockpit, and is external to the synthesis and logic design process. So while a determined logic designer can quickly generate a physical prototype if he has all the physical library and setup info he needs, it will not provide to him the type of feedback that he needs to affect changes in his realm.

First Encounter silicon virtual prototyping has for a number of years provided physical designers with a quick way to do accurate chip-level prototyping. It uses legal placement and trial routing, which is basically detailed routing minus all the long search-and-repair to legalize it. It's enough to get us the capacitance we need. It has been the “de facto standard” SVP solution, used as the “front-end to the back-end process”, whether the back-end is Cadence or not.

# RC Built-In Physical Capabilities

---

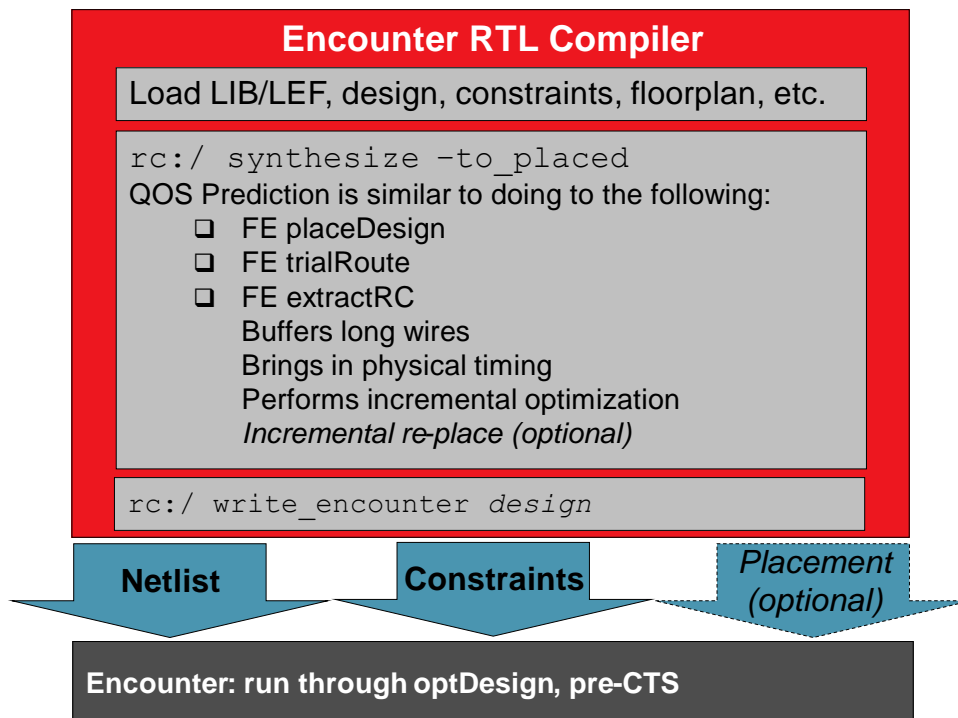
- ◆ **Premorphing** – RCP uses premorphing to reduce local utilization congestion. The idea is to move cells from highly utilized regions to low utilization regions using flow based algorithms.
- ◆ **Incremental placement** – RCP automatically places any unplaced gates generated during synthesis. In general, all IOPT tricks ensure that any gate that is changed or newly created gets placed.
- ◆ **Legalization** – RCP legalizes the placement automatically.
- ◆ **Native Congestion Estimation & Optimization** - Available track calculation per metal layer, accounts for routing blockage and power routes. All metal layer 1 routing blockages are treated as placement blockages automatically.

`synthesize -to_placed`

`synthesize -to_placed -incremental`

- ◆ RCP can read/write DEF – Output a fully placed, legal design

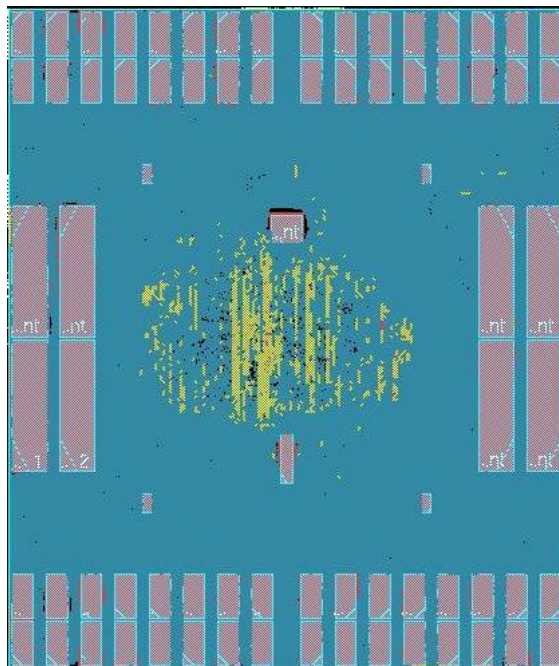
# The RC Physical Process



# Automatic Congestion Fixing

When running physical synthesis, the software automatically does the following:

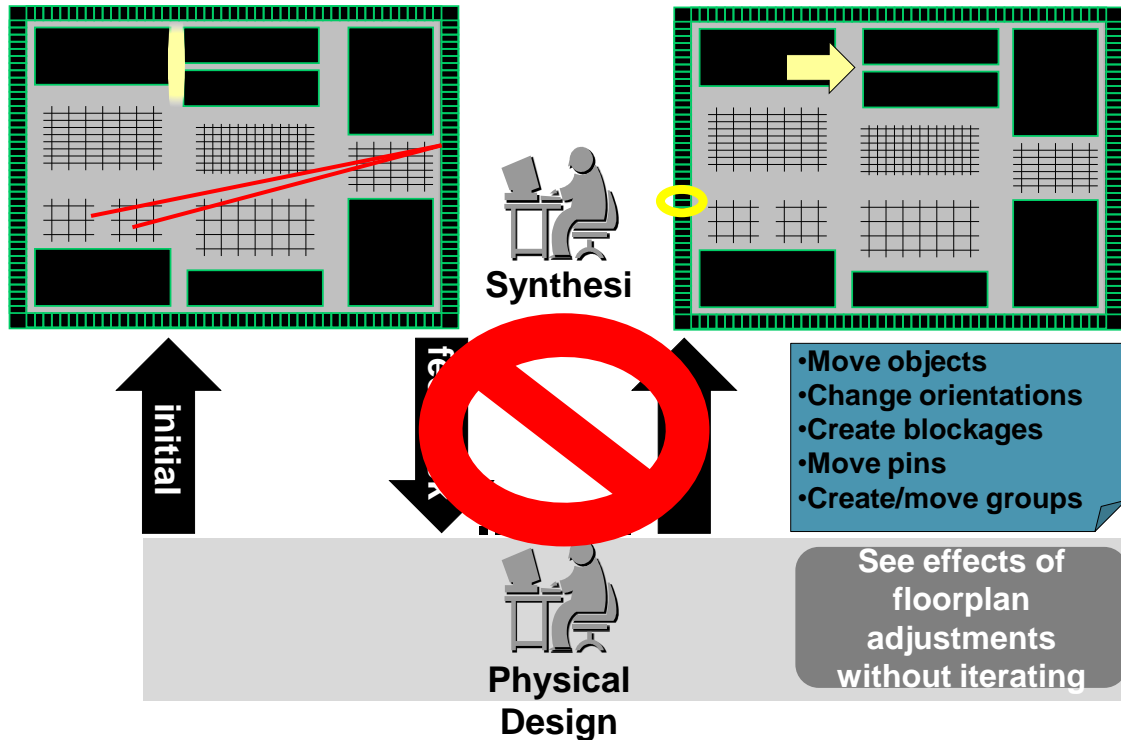
- ◆ Native congestion estimation and reporting
- ◆ Enhanced morphing to relieve congestion hot-spots
- ◆ Global whitespace redistribution
  - ❑ Individual gate pin-density
  - ❑ Local target utilization
  - ❑ Global interconnect
- ◆ Congestion-aware incremental optimization



**After**

This requires an *RTL\_Compiler\_Physical* license.

# Incremental Floorplan What-if Tweaking



04/25/11

Encounter RTL Compiler

275

RC-Physical can also help identify floorplan issues, like wires that have to cross too much of the chip, macro placement that causes congestion, the need to create blockages to account for blocks not yet available, etc.

Bothering the physical team to make small changes to the floorplan, wait to get a new floorplan, see the results, possibly ask for more adjustments, etc. can waste days or even weeks for each pass

We have added some commands to perform small what-if adjustments to the floorplan while in RC-Physical, so the synthesis user can try these out without bothering physical design and having to wait for new passes. The physical team will ultimately need to fix the final floorplan on their side.

# write\_template -split -physical -outfile run.tcl

---

## setup\_run.tcl:

```
## Include leakage and dynamic power inQoS reporting
set_attributeqos_report_powertrue /
set_attributeenc_gzip_interface_filetrue /
set env(ENCOUNTER) <Encounter executable path>
regexp [0-9]+(\.[0-9]+) [get_attributeprogram_version/] exe_verexe_sub_ver
puts "Executable Version: $exe_ver"
```

## run.tcl:

```
synthesize-to_mapped-eff $MAP_EFF-no_incr
puts "Runtime & Memory after 'synthesize-to_map-no_incr'"
generate_reports-outdir $_REPORTS_PATH-tag map
summary_table-outdir $_REPORTS_PATH
write_design-encounter-gzip-basename $_OUTPUTS_PATH}/map/${DESIGN}

#####
## QoS Prediction & Optimization.
#####
set_attributeenc_temp_dir$_OUTPUTS_PATH}/c_enc_pred/
synthesize-to_placed-effort $PHYS_EFF
# generate reports to save the encounter stats
generate_reports-outdir $_REPORTS_PATH-tag plc_enc-encounter
summary_table-outdir $_REPORTS_PATH
write_design-encounter-gzip-basename $_OUTPUTS_PATH}/plc/${DESIGN}

#####
## Final: write Encounter file set (verilog, SDC, config, etc.)
#####
generate_reports-outdir $_REPORTS_PATH-tag final
summary_table-outdir $_REPORTS_PATH
write_design-encounter-gzip-basename $_OUTPUTS_PATH}/final/${DESIGN}
```

04/25/11

Encounter RTL Compiler

276



# Advanced Synthesis Features

## Appendix B

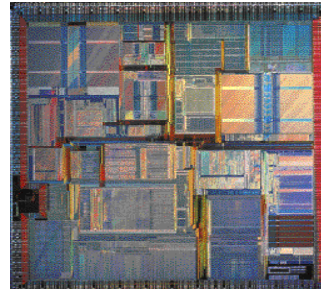
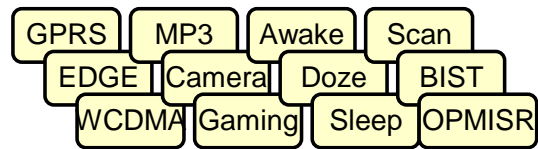


April 25, 2011

# Multiple Mode Design

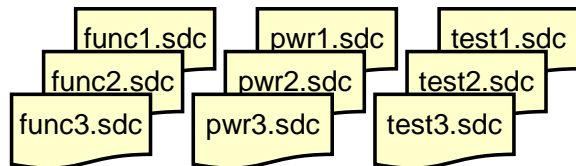
Today's chips include:

- ◆ 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?

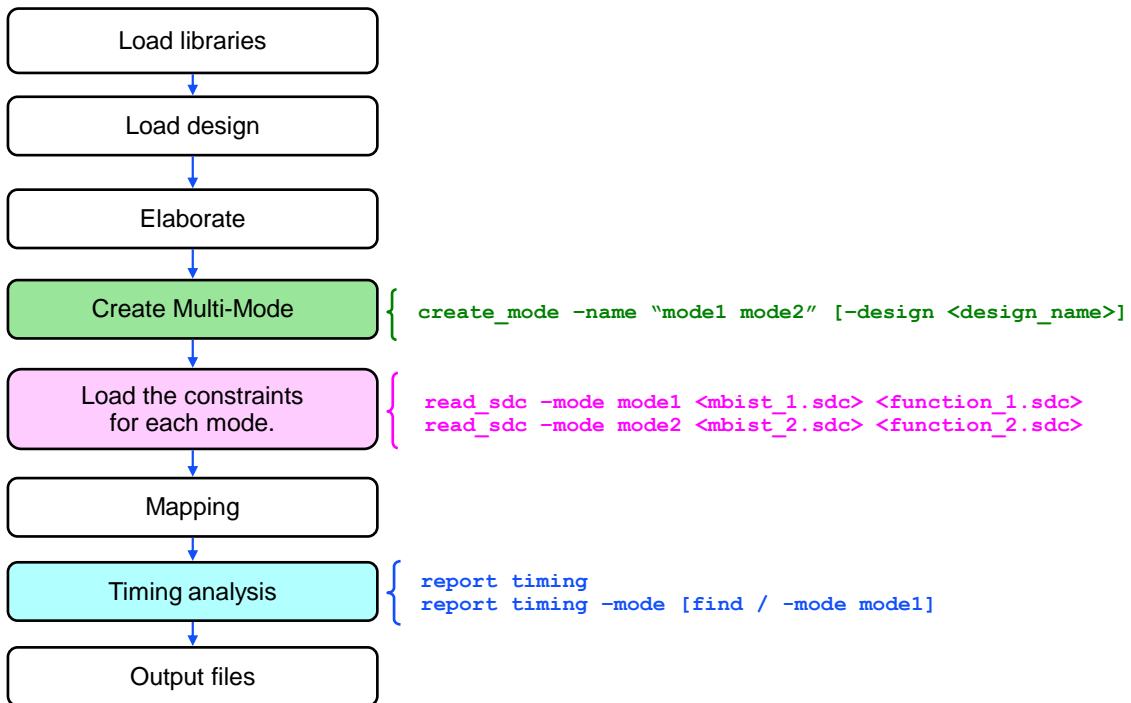


Chip design today is becoming very complicated due to integration. We constantly see announcements touting single-chip solutions that support multiple standards, or that perform multiple functions. In addition to today's power densities, chips operate in different modes to conserve power. And where there 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 Flow

---



04/25/11

Encounter RTL Compiler

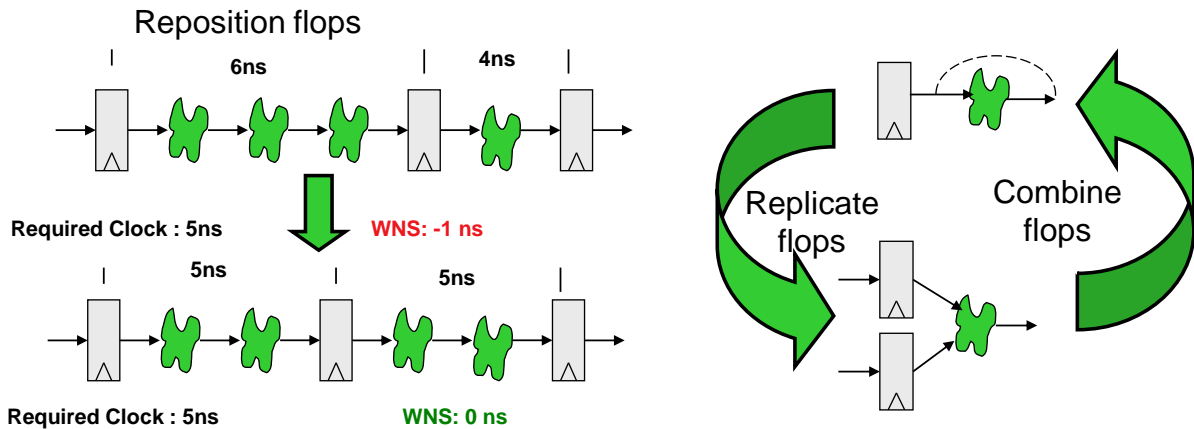
279

If you are using the multi-mode flow, then you must specify the `-mode` option to each of the commands.

# Retiming

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 des seqs -clock clk2]
```



04/25/11

Encounter RTL Compiler

280

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 run time.

```
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\_l\_flop* instead of the default, *retime\_l\_reg*.

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

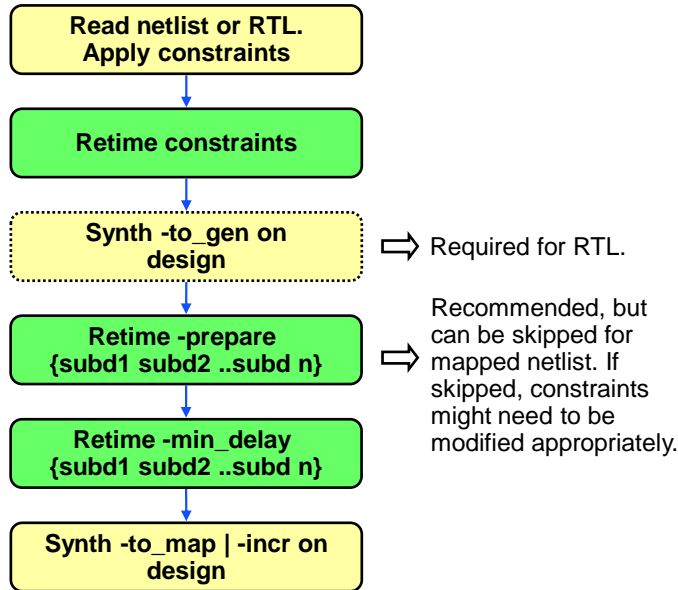
Select flops with *trace\_retime true* for tracing the original name of the flops. Use on limited flops only. To get the status of these flops, use:

```
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]
```



04/25/11

Encounter RTL Compiler

281

## 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.

# Deriving Environment

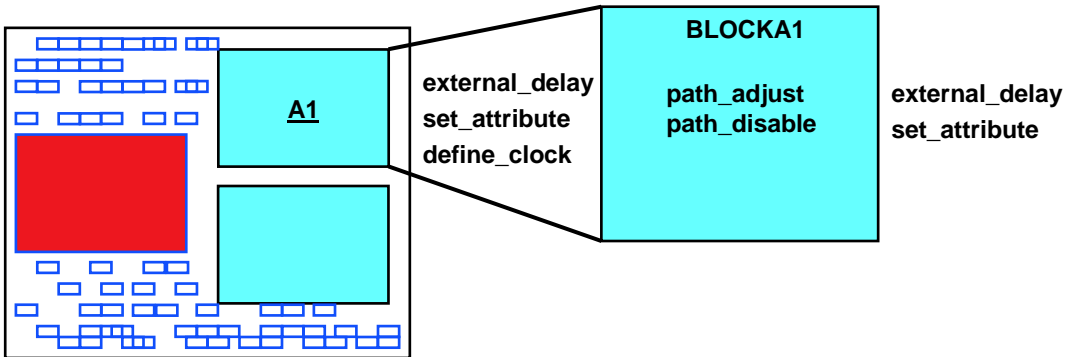
Use the *derive\_environment* command to extract the timing constraints for a subdesign based on the timing constraints it had in the original design. The slack at each pin in the extracted design matches the slack at the corresponding pin in the original design.

## Syntax

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

## Example

```
derive_environment A1 -name BLOCKA1
```



04/25/11

Encounter RTL Compiler

282

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.
- ◆ Map the top level.
- ◆ Derive constraints for the block.
- ◆ Synthesize the block with a high effort.
- ◆ Go to the top level and link the new block.
- ◆ Reapply any multicycle constraints, because they might be removed during *change\_link*.
- ◆ Preserve instances as needed.
- ◆ Resynthesize the top level with the proper effort level.
- ◆ Report, analyze, and save.

```
read_hdl $FILE_LIST
elaborate
```

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

```
derive_environment -name dma
                   DTMF INST/DMA INST
```

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

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

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

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

---

Blank Page



# Encounter RTL Compiler Constraints

## Appendix C



April 25, 2011

# 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 Compiler Tcl Constraints

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

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).

# 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 <i>fanin</i> command.
Multiple clocks propagating to the same sequential clock pin	To see which clocks are being propagated to that pin, use the <i>inverting_clocks</i> or <i>non_inverting_clocks</i> attribute of the pin. Use the <i>timing_case_logic_value</i> attribute to propagate only one clock to that pin ( <i>set_case_analysis</i> ).
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.

04/25/11

Encounter RTL Compiler

288

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

# Defining a Clock

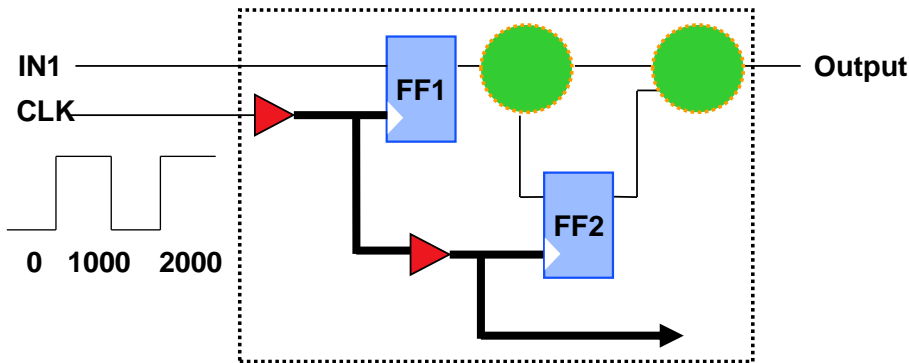
Use the `create_clock` SDC command to define clock objects and their associated details such as a clock waveform.

```
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



04/25/11

Encounter RTL Compiler

289

## 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 <b>pico</b> seconds
-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 Uncertainty

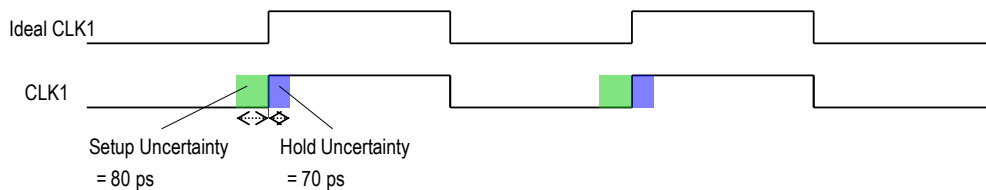
Set the clock uncertainty using the following SDC command:

```
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]
```



- In a single clock domain, uncertainty specifies setup or hold uncertainties to all paths to the endpoint.
- Inter-clock 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.

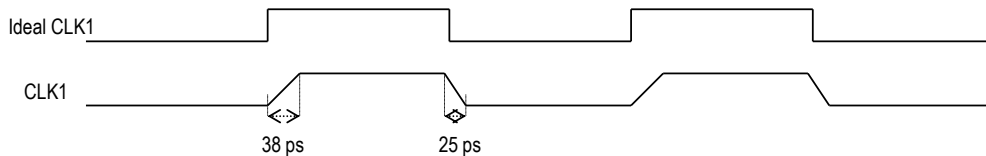
# Setting Clock Slew

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]
```



04/25/11

Encounter RTL Compiler

291

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.

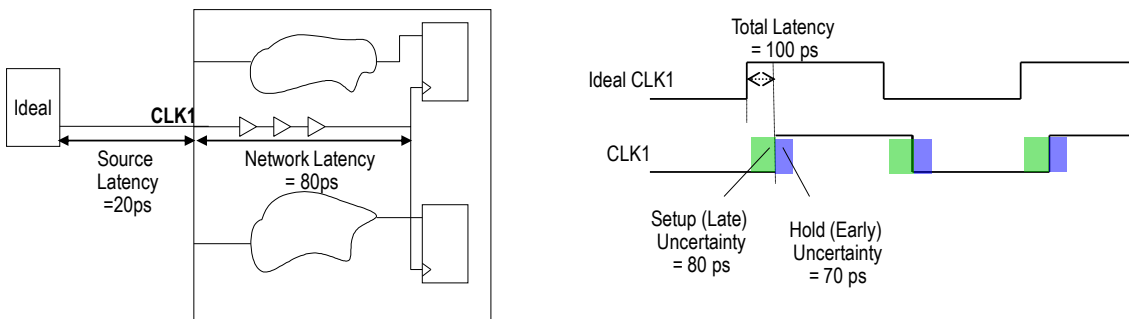
# Modeling Clock Latency

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]
```



04/25/11

Encounter RTL Compiler

292

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

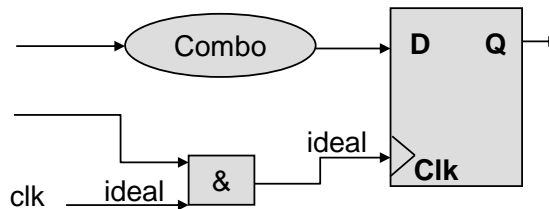
Use the following SDC command to specify which nets are ideal.

```
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.



04/25/11

Encounter RTL Compiler

294

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.

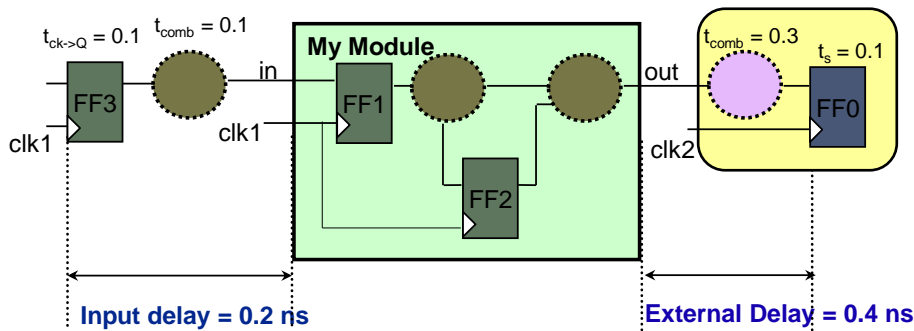
# Input and Output Delays

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]
```



04/25/11

Encounter RTL Compiler

295

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.

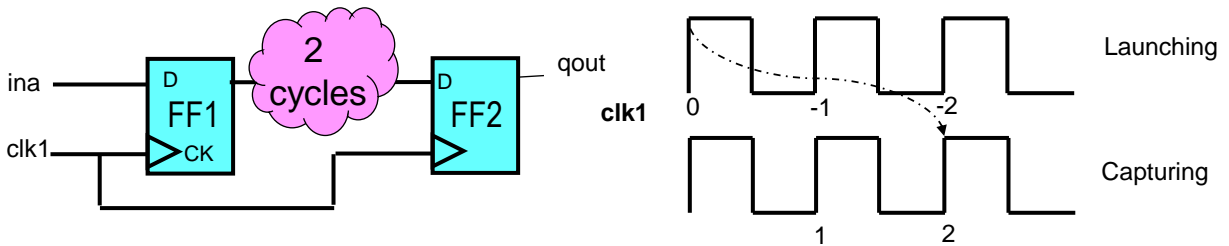
## Example

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 mcpl
```



04/25/11

Encounter RTL Compiler

296

The smallest positive difference between launching clock edge and capturing clock edge is the default timing constraint.

# Setting False Paths

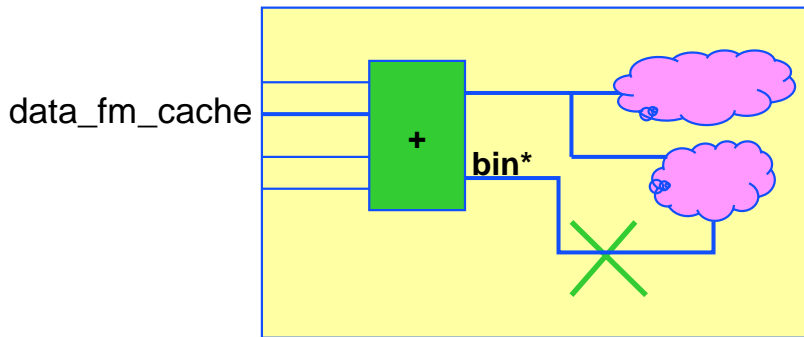
Set false paths using the following SDC command:

## 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
```



04/25/11

Encounter RTL Compiler

297

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.

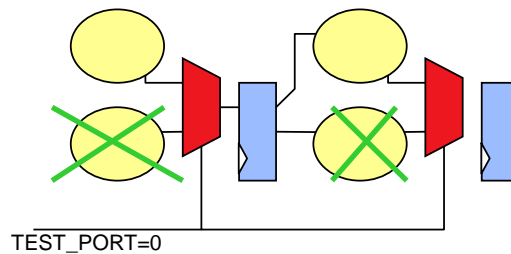
## Example

```
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*.



04/25/11

## Encounter RTL Compiler

300

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 *therm* 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>
```

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

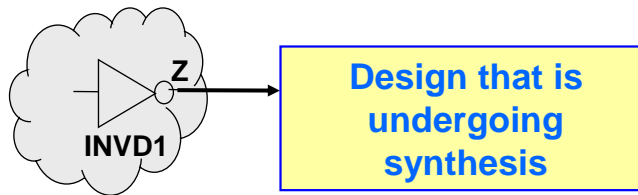
---

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/*
```



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]
```

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 C Applying RTL Compiler Constraints (Optional)

- Setting Up the Environment
- Setting Clocks
- Applying External Delays
- Setting Ideal Drivers
- Applying Path Exceptions
- Setting Design Rule Checks