ECE 524-Synthesis & verification of digital circuits
Spring 2015

Cadence - Encounter RTL Compiler Tutorial

Instructor: Dr. Spyros Tragoudas
Office: E0202b
Email: spyros@engr.siu.edu

Teaching Assistant: Wisam Aljubouri
Office: A119
Email: wisam@siu.edu

Department of ECE
Southern Illinois University Carbondale

1 Introduction

This Tutorial aims in giving a brief introduction on how to use Cadence Encounter RTL Compiler(RC) to perform design synthesis. **This is neither a full tutorial, nor a full manual about synthesis using the Encounter RTL Compiler**. The user manual of this tool is available in the path **/cadence/RC/doc/rc_user/rc_user.pdf**. Actually this user manual is one of the available manual under cadence directory.

1.1 Brief Introduction About Synthesis

Logic synthesis is the process which converts a design written at the Register Transfer Level *(RTL)* into a gate level net-list. The RTL specification is written in Verilog or VHDL, using high-level constructs such as for loops and case statements. The synthesis tool transforms this RTL specification into a set of logic gates, such as AND, OR, and BUF, that are connected to together to form the net-list. To specify the gates used by the synthesis tool to build a net-list, a technology is chosen from a specific vendor. The vendor supplies a technology library for the synthesis procedure. The technology library defines the physical properties of the gates, including the amount of time that is required for a signal to pass through each gate. In addition to creating a gate level net-list, the synthesis tool can perform the following functions.

1. Analyze the timing of the net-list to ensure that no timing errors can occur.
2. Optimize the design for either the best performance or the smallest size.
3. Automatically insert a chain of scan elements or test signals into the net-list.

1.2 Brief Introduction About Encounter RTL Compiler

Encounter RTL Compiler is used to generate optimized gate level net-lists from the RTL models, in the following order

1. Read the technology library into the synthesis database.
2. Read the HDL source code for the design, written in Verilog or VHDL, into the synthesis database.
3. Generate a generic net-list based on the generic library.
4. Map the generic net-list to cells in the technology library.
5. Optimization of the design under different design constraints.

2 Logic Synthesis using Cadence Encounter RTL Compiler

1. Before starting the simulation, make sure that you are in correct directory.

   In this tutorial, the current directory is **/grad/aljubouri/ECE524/RC**.

2. To initialize cadence,

   Type: **cadence** and click **enter** in the terminal.

   The terminal is changed to cadence mode.

   Note: Cadence should be initialized before using cadence tools, whenever a new terminal is opened.

3. To check whether the Cadence Encounter RTL Compiler is properly setup or not,

   Type: **which rc** and click **enter**.

   If the terminal shows the path as **/cadence/RC/tools/bin/rc**, then Encounter RTL Compiler is properly setup and ready for use.

4. In order to know how to work on Encounter RTL Compiler, now copy a small VHDL source file **FullAdder.vhd** into your RC directory.

2.1 To invoke Encounter RTL Compiler

Encounter RTL Compiler has both the command-line interface and a graphical user interface (GUI). Both provide the same synthesis functions. The tool takes behavioral description of Verilog and VHDL designs and synthesizes it to a gate level net-list.

To launch the Encounter RTL Compiler in the terminal, Type: **rc -gui** and click **enter**

Note: Don't use & in the end of the command.

This command starts RTL Compiler Ultra version 9.1.1(GUI) as shown in Figure 1. In the terminal, the path will be changed to **rc:/>**

The window has two areas:

1. Source code editor(HDL)(left) gives access to the HDL source files. Any changes made in the source files can generate a new net-list from those changes.

2. Schematic viewer (right) displays the design in schematic form. It allows to pan and zoom, display fan-in and fan-out cones, and display critical paths and timing values. It also allows to group instances, dissolve instances, or change the reference point of an instance. Unlike other GUI interfaces, this GUI is the initial window from which **rc** was launched that why it had to be launched in the foreground with out the *ampersand*.

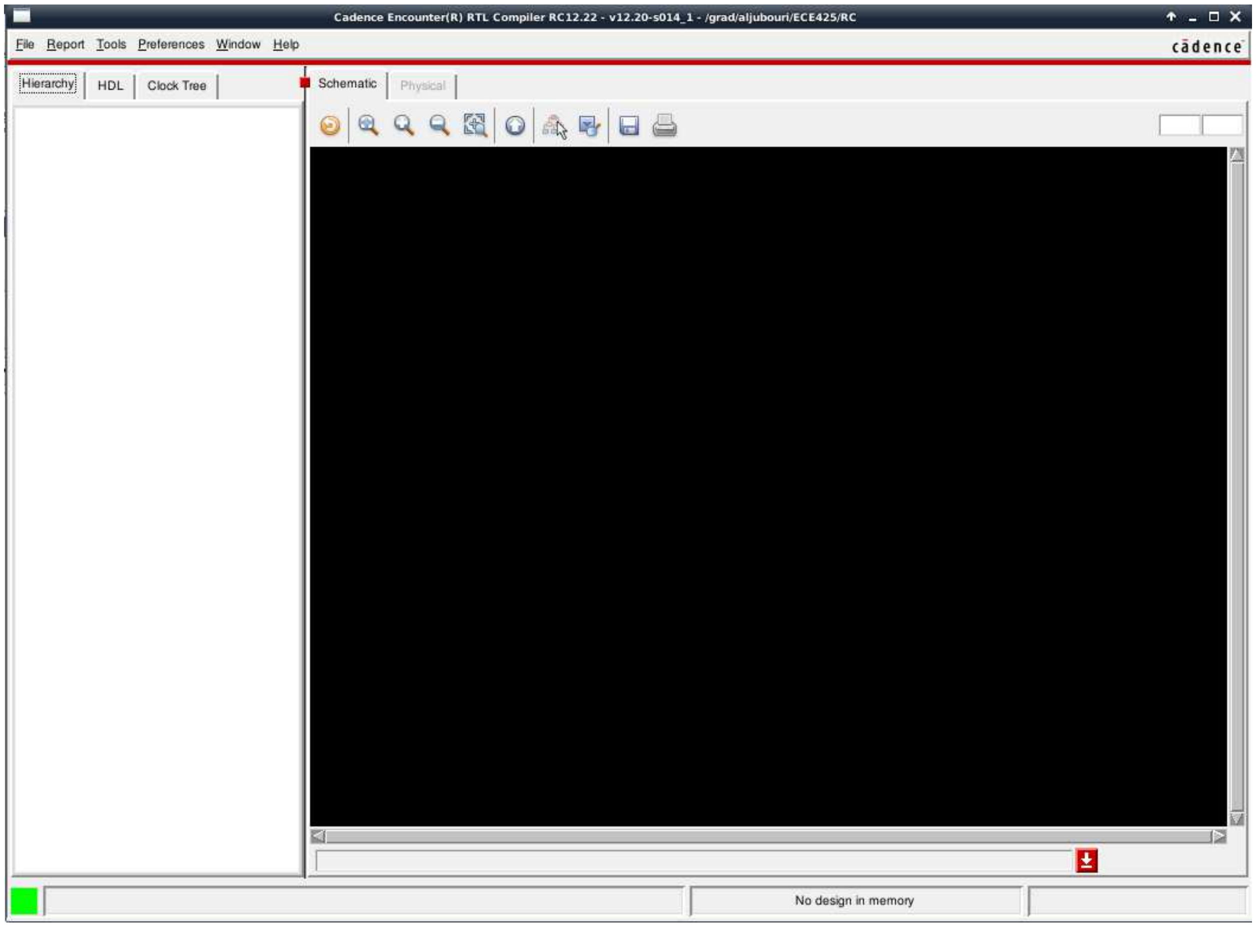

Cadence Encounter(R) RTL Compiler RC12.22 - v12.20-s014_1 - /grad/aljubouri/ECE425/RC
File Report Tools Preferences Window Help
cadence
Hierarchy
HDL
Clock Tree
Schematic
Physical
No design in memory

Figure 1. Initial Gui

2.2 Setting the library of standard cells

There are many technology libraries available in cadence. In this course, we use **Artisan TSMC 0.18m** physical and timing libraries.

To set the search path for this tool, Type

set_attribute lib_search_path /cadence/GPDK/osu_stdcells/lib/tsmc018/signalstorm and click **enter**.

osu018 stdcells.lib is a description of the TCMS standard cell library with 0.18 micro technology. In this course, we are using this technology library of cadence. To set the technology library

Type: **set_attribute library osu018_stdcells.lib** and click **enter**.

These two commands will set search path and the library of standard cells to be used with the design.

set_attribute is the command used in **rc**. To know more details on this command, Type: **man set_attribute** and click **enter**.

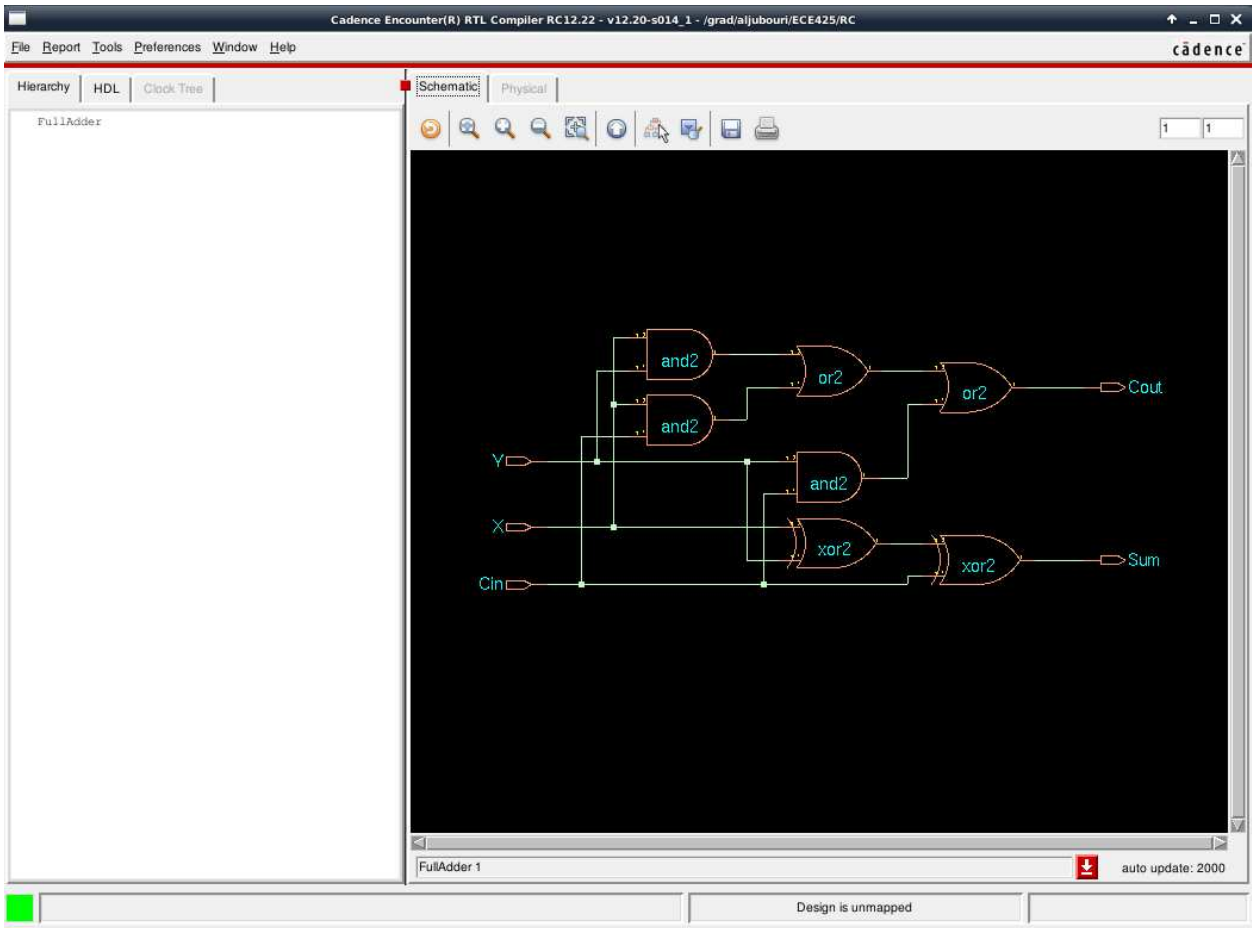


Figure 2. Unmapped Design

2.3 Loading the HDL file

Encounter RTL Compiler is not a HDL code compiler, so any errors in the HDL code will not be verified by it. Make sure the HDL code is *error free* and *synthesizable*. The HDL codes should be compiled and simulated using NClaunch or any other HDL simulators, to ensure that they are *error free*. The code which contains any time information is referred as *not synthesizable*, in other words all the variables of type *TIME* and all the *after* statements should be removed before the loading the HDL file into Encounter RTL Compiler. The example file, *FullAdder.vhd* is an *error free* and *synthesizable* code.

To load VHDL file, Type: **read_hdl -vhdl filename.vhd** and click **enter**.

To load Verilog file, Type: **read_hdl filename.v** and click **enter**.

In this example to load *FullAdder.vhd*,

Type: **read_hdl -vhdl FullAdder.vhd** and click **enter**.

This command will load the *FullAdder.vhd* file into **rc**. If there is any errors, the terminal will show the errors and warnings.

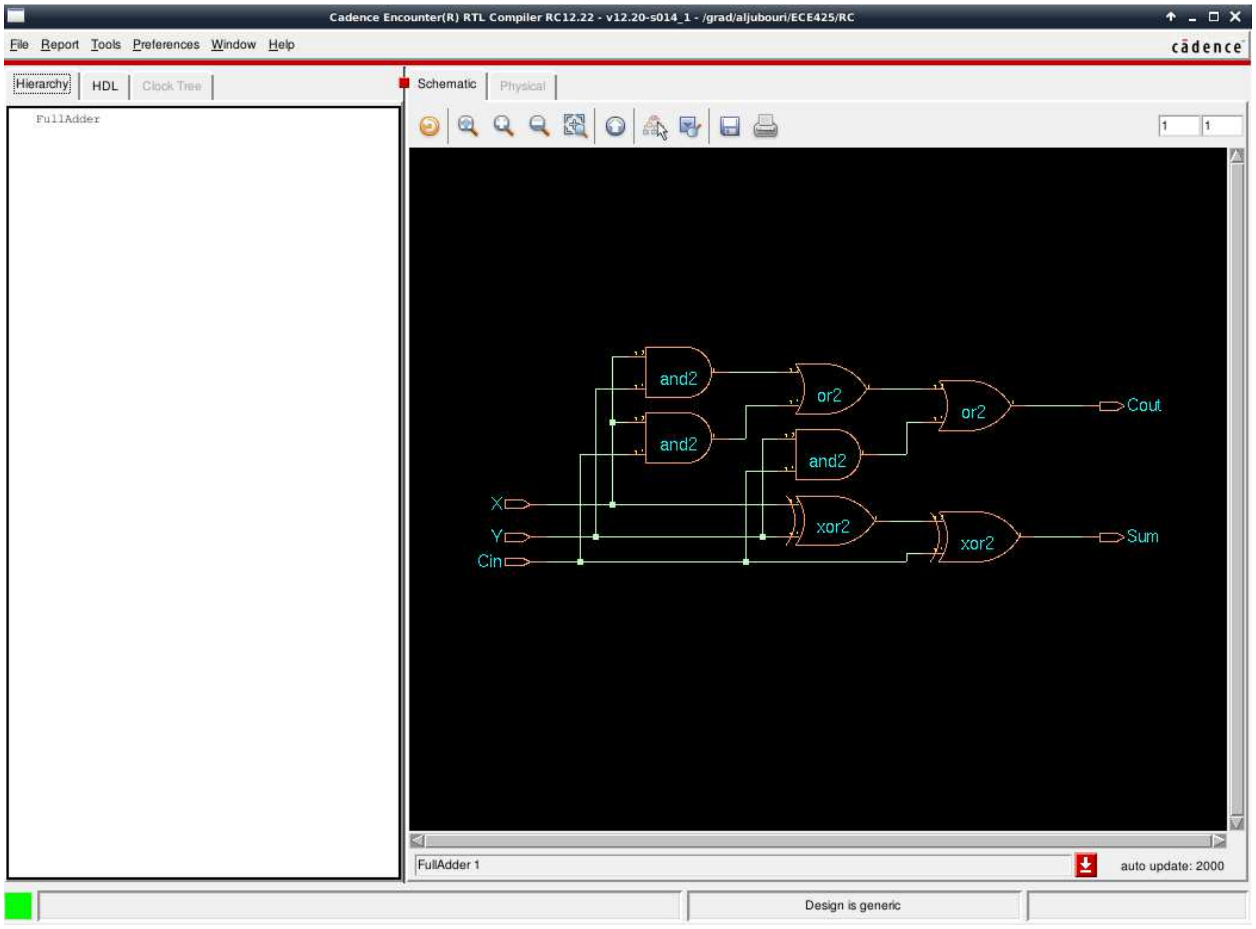


Figure 3. Generic Design

2.4 Elaboration

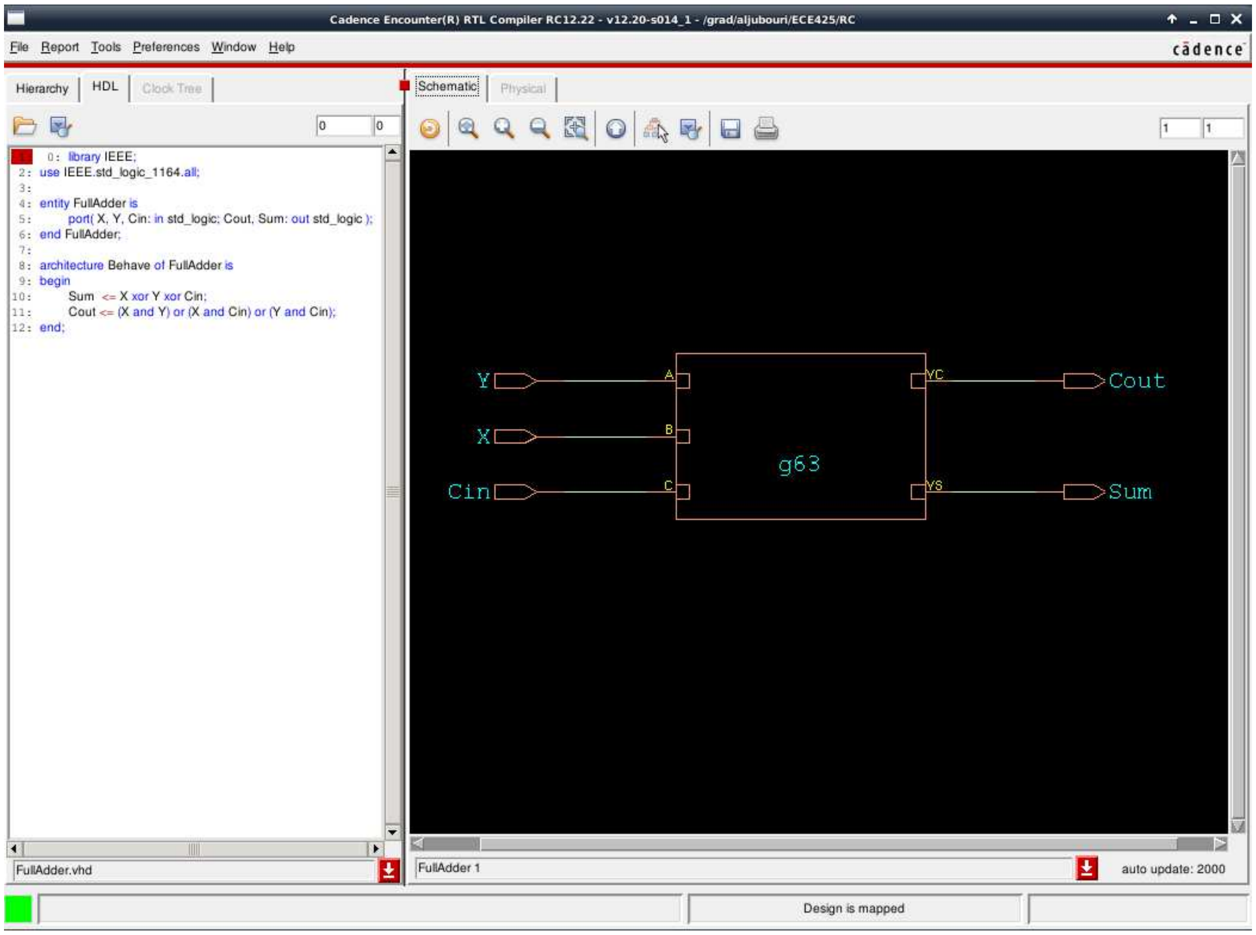
Elaboration is the step to convert the design from the HDL description into a Cadence specific format that is required for synthesis.
For Elaboration, Type: **elaborate** and click **enter**.

After elaborating the net-list of the design can be seen in the GUI window like as shown in Figure 2. Also the GUI window allows to freely navigate through the net-list, by zooming in and out, selecting components or even inspect designs at lower levels by double-clicking on them.

2.5 Synthesize

Now the Full Adder circuit is ready for synthesize. To do so
Type: **synthesize -to_generic** and click **enter**.

After the above command the circuit is the same as before, because elaboration is essentially a synthesis process with generic library cells. The only difference between Figure 2 and Figure 3, the bottom



```
library IEEE;
use IEEE.std_logic_1164.all;

entity FullAdder is
        port( X, Y, Cin: in std_logic; Cout, Sum: out std_logic );
end FullAdder;

architecture Behave of FullAdder is
begin
        Sum  <= X xor Y xor Cin;
        Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end;
```

Figure 4. Mapped Design

window message is different, where in Figure 2 indicates the design is unmapped. Despite the fact that a synthesis succession message appears in the console, the synthesis process continues for some more time. Wait until no progress bar is active in the GUI window as shown in Figure 3. This indicates design is generic.

To do the mapping the design to specified technology library ,

Type: **synthesize -to_mapped** and click **enter**.

The design will be mapped to the specified technology defined in the library attribute that was set before. Now the net-list of selected component will be totally different as shown in Figure 4, since the technology mapping found a *FullAdder* component in the standard cell library.

The above mentioned steps are standard procedures to load the HDL files, synthesize and map to particular technology Library.

2.6 Export the results

The synthesized design can be exported after this synthesis process in a Verilog description (which is suitable for importing the design to other tools). For the given example,

For the generic mapped description,

Type: **write_hdl -generic FullAdder > FullAdderGen.v** and click **enter**.

For the technology mapped description,

Type: **write_hdl -mapped FullAdder > FullAdderMap.v** and click **enter**.

Note: To know more about the RC commands and their usage click *Help → Man Page Browser*.

2.7 Inspection of Results, Redirecting Reports and Removing the Designs

The RTL Compiler GUI, visualizes some of the various reporting and design analyzing features of the tool. Start by clicking the *Tools → Object Browser*. A new windows will pop up that reports all the designs that are active in the current instance of the tool, the HDL libraries and the user defined libraries. Take some time to explore the various reported values and information. Next, click on *Report*. Here, the information about the design will be available . Start with the option *Netlist → Statistics* and *Netlist → Area*. Here the information regarding the area occupied by the design, the logic used or other components can be seen. The second option provides analytical area information per component.

The report command sends the output to stdout by default; you can redirect stdout information to a file or variable with the redirect command.

To write the *area report* for the *FullAdder* to a file called *areaFA.rep*

Type, **report area > areaFA.rep** or **redirect areaFA.rep "report area"** and click **enter**

Use **Man Page Browser** to see all the possible reports to be produced by this tool.

If you want to remove any designs from your *rc* memory.

Type: **rm *design name*** and click **enter**. **Note: *design name* is the entity or module name in the HDL file**.

To delete all the previous compiled designs located in the *rc* memory.

Type, **rm /designs/*** and click **enter**.

Generally, when you close the **rc** and all design are deleted automatically. When you load the same code more than once in **rc**, it will store it as different design. For example, after synthesize the **FullAdder**, if you once again load and synthesize the *FullAdder*, it will stored as a different design *(example: FullAdder_1)*

2.8 Large Designs

For larger designs with many HDL files, the procedure of loading and synthesize procedure from the command line becomes very time-consuming and error-prone. The RTL Compiler provides an easy and efficient way of combining commands into TCL files. An example file, called *FullAdder.tcl* has been given. These lines are nothing more than all the commands given abov. Go through the commands in this *tcl* file, before executing. To execute the file, go to the GUI window and click on menu File → Source Script. Then select the *FullAdder.tcl* file and click OK. This will performed all the steps described above.

3 Area Optimization Using Encounter RTL Compiler

Encounter RTL Compiler is used for optimizing design parameters such as area, timing, and power of the given design. This section aims in giving a brief introduction on how to use Cadence Encounter RTL Compiler to perform simple area optimization actions. **This is neither a full tutorial, nor a full manual about area optimization using the Encounter RTL Compiler**. Area Optimization using Encounter RTL Compiler can be done in many ways. Here we discuss using two simple commands for area optimization.

By default, RTL Compiler tries to fix all *DRC* errors, but not at the expense of timing. If *DRCs* are not being fixed, it could lead infeasible slew issues on input ports or infeasible loads on output ports. So it is necessary to force RTL Compiler to fix *DRCs*, even at the expense of timing during any optimization.

To ensure *DRC* get solved, Type:
set_attribute drc_first true and click **enter**.

By default, this attribute is false, which means that DRCs will not be fixed if it introduces timing violations.

3.1 Including and/or Excluding Logic Components

Technology library has more than one *libcells* for implementing same logic component. In Figure 4, *libcell:FAX1* is one of the *libcell* for the *FullAdder* located in the library osu018. Each *libcell* for same logic complement differs in many of its design parameters. Choosing among these *libcells* for same logic complements is determined by the optimization objective. To synthesize and map a design, **rc** uses some default *libcell*s.

We can exclude the usage of a specific *libcell* from the synthesis procedure using the command below.

set_attribute avoid true *libcell* and click **enter**.

Example : set_attribute avoid true FAX1 and click **enter**.

This command asks the RTL compiler not to include the *FAX1* cell in the synthesis procedure. i.e avoid using FAX1 cell located in the standard cells library named osu018_stdcells.

To remove above the exclusion constraint,

Type, **set_attribute avoid false *libcell*** and click **enter**.

Example: set_attribute avoid false FAX1 and click **enter**.

Whenever, you use any of the above two commands to observe the change in the design, once again Type,
synthesize -to_mapped and click **enter**.

3.2 Using effort option

For small designs, it easy to chose the *libcell* for including or excluding for area optimization. However, it is difficult for large design which contains millions of different *libcell*s. For this, we use option *effort* during mapping procedure, where RTL maps the *libcell*s according to the *effort* option. Use the *man* page to know how to use the *effort* option along with the **synthesize -to_mapped** command.