

## Laboratory 5

- *Using Design Compiler for Synthesis*

By Mulong Li, 2013

Reference:

[http://www.tkt.cs.tut.fi/tools/public/tutorials/synopsys/design\\_compiler/gsd.html](http://www.tkt.cs.tut.fi/tools/public/tutorials/synopsys/design_compiler/gsd.html)

## Background knowledge

**Synthesis:** is the process of taking a behavioral HDL file and converting it to a structural file using cells from a standard library. There are many different tools available in the CAD market targeting ASICs, among which, *Design Compiler* from Synopsys is used by most engineers. (Don't get confused with ISE or Quartus II, which are synthesis tools targeting FPGAs.) We'll use *Design Compiler* in this tutorial.

**Design Compiler:** is the flagship synthesis tool from Synopsys. It comes in two flavors: *dc\_shell*, which only has Tcl shell-command interface, and *Design Vision*, which is a GUI window/menu driven version.

**HDL:** Hardware Description Language is a specialized computer language used to describe the structure, design and operation of electronic circuits. Verilog and VHDL are the most widely-used and well-supported HDLs.

# 1 Getting Started

In this tutorial, we will synthesize a 4-bit ripple carry adder. The VHDL codes will be provided, so you don't have to worry about writing them.

## 1.1 Synopsys Design Compiler setup:

- a) In your home directory, type “synopsys-cme342” to start Synopsys DC Directory  
~/cabinet/CME342/synopsys\_dc/ will be automatically created for you.

**\$ synopsys-cme342**

- b) After a while you should see a message similar to the one below indicating your environment is set up correctly:

DC Professional (TM)  
DC Expert (TM)  
DC Ultra (TM)  
FloorPlan Manager (TM)  
HDL Compiler (TM)  
VHDL Compiler (TM)  
Library Compiler (TM)  
DesignWare Developer (TM)  
DFT Compiler (TM)  
BSD Compiler  
Power Compiler (TM)

Version D-2010.03-SP4 for linux -- Aug 27, 2010  
Copyright (c) 1988-2010 by Synopsys, Inc.  
ALL RIGHTS RESERVED

This software and the associated documentation are confidential and proprietary to Synopsys, Inc. Your use or disclosure of this software is subject to the terms and conditions of a written license agreement between you, or your company, and Synopsys, Inc.

The above trademark notice does not imply that you are licensed to use all of the listed products. You are licensed to use only those products for which you have lawfully obtained a valid license key.

Initializing...  
loaded paths for 180nm.....

- c) In order to keep your project data well-organized, a few files and sub-directories has been created in the following structure:

```
synopsys_dc/  -- project directory
  .synopsys_de.setup -- Synopsys Design Compiler initialization file, do not change it
  SRC/        -- HDL source files
  SYN/        -- Synthesis subdirectory
    DDC/      -- Design Compiler database
    NETLIST/  -- Mapped HDL netlists
    RPT/      -- Reports
    SCP/      -- Synthesis scripts
    WORK/     -- Intermediate files from synthesis tool
  SIM/        -- Simulation subdirectory
    SCR/      -- Simulation scripts
    WORK/     -- ModelSim work directory
```

- d) Download the 3 VHDL files: *ha.vhd*, *fa.vhd*, *rca.vhd* from the course website, and put them into the directory SRC/.

## 1.2 Ripple Carry Adder (RCA)

In this lab, we will take a 4-bit Ripple Carry Adder (RCA) as an example to illustrate the basic synthesis flow: reading the design, setting constraints, optimizing the design, reporting and analyzing, and saving the design.

### a) Half Adder

The half adder adds 2 single binary digits A and B. It has 2 outputs, sum(S) and carry(C).

Figure 1.1 shows the gate-level structure of the Half Adder (HA) module, which incorporates an XOR gate for S and an AND gate for C. This design is described in file *ha.vhd*.

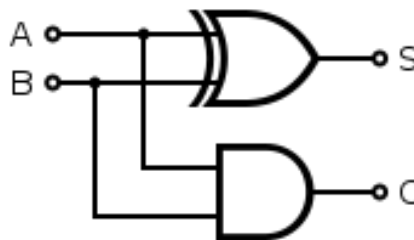


Figure 1.1

### b) Full Adder

A full adder adds binary numbers and accounts for values carried in as well as out. The Full Adder (FA) module is composed of two instances of Half Adders and an OR gate as shown in Figure 1.2. The design is in file *fa.vhd*.

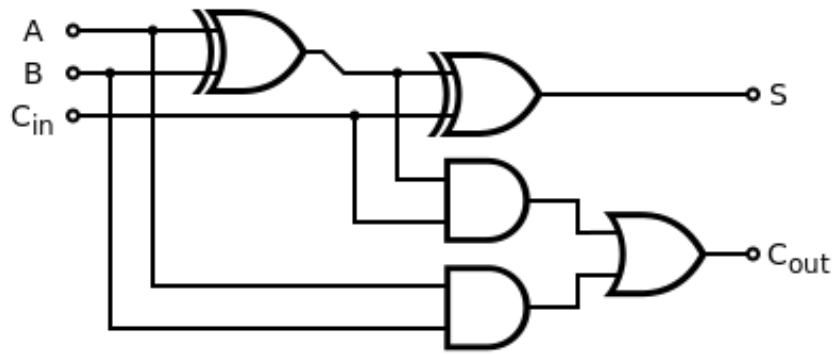


Figure 1.2

### c) Ripple Carry Adder

The 4-bit RCA design is instantiation of 4 Full Adder Modules as shown in Figure 1.3. The design is described in file *rca.vhd*.

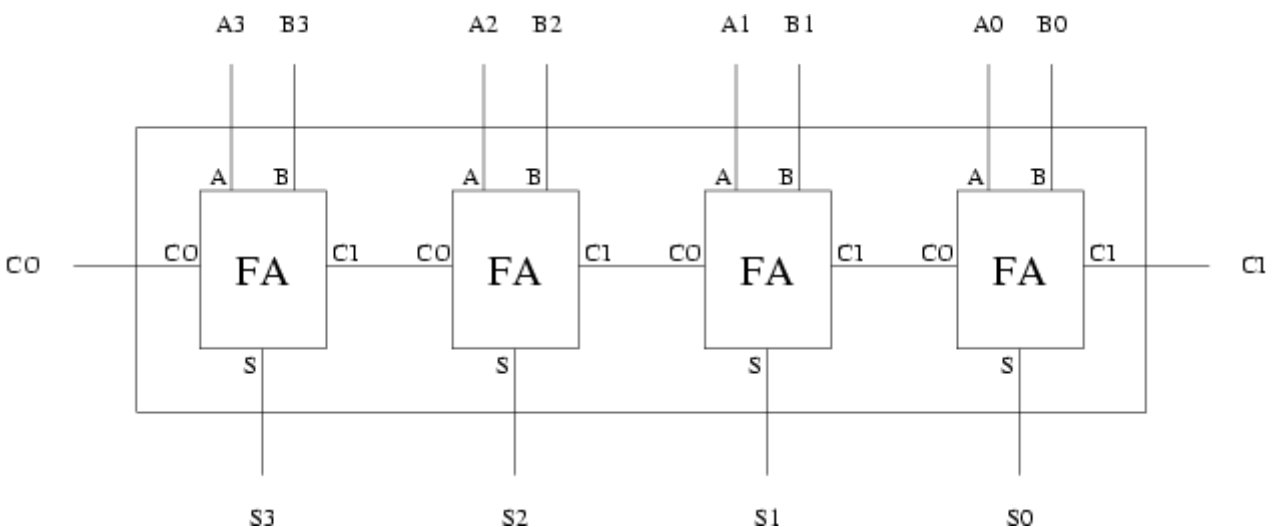


Figure 1.3

## 1.3 Design Vision

- a) Design Vision is the GUI for Design Compiler. Invoke Design Vision with command “gui\_start” in dc\_shell. In a moment, you should see a window similar to Figure 1.4.

```
$ gui_start
```

- b) The Hierarchy Browser displays information about the design in textual form. The Logical Hierarchy panel displays the design's hierarchy and the objects list on the right displays information about the objects in current instance.

The Console views display commands entered by the designer and messages resulting the commands. Commands can be entered in Console Command Line.

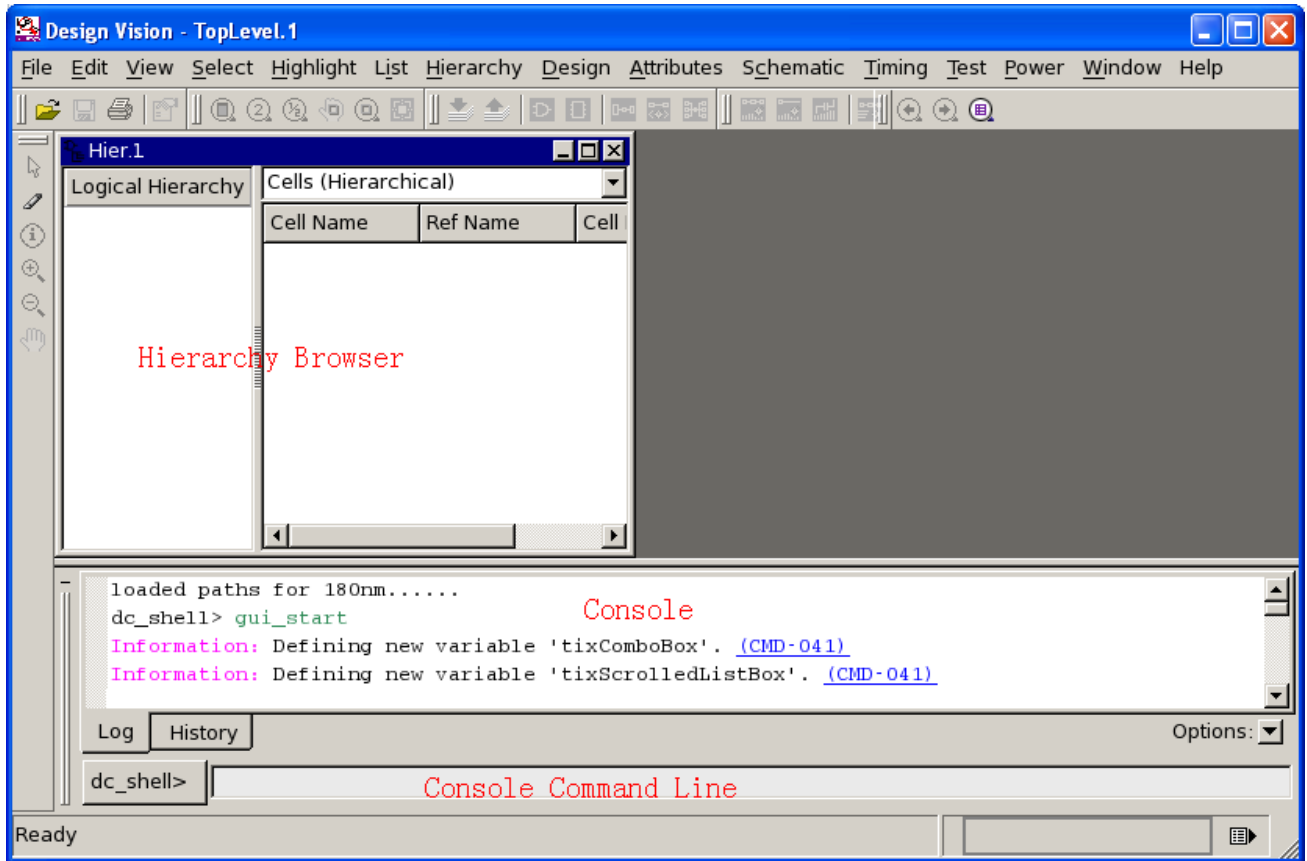


Figure 1.4

- c) **Only some of the most common commands can be performed by using the Design Vision GUI menus and buttons.** All commands can always be executed by writing them into the console. In this tutorial, you can choose to either write the commands into the console or using the menus.
- d) Go to Help -> Man Pages, you'll see the "Man Page View", which gives more detailed description of all the commands, variables and messages.
- e) Note: **there's no "undo" in Design Compiler.** Make sure you are doing it right before you hit Enter!

## 2 Synthesis Flow

### 2.1 Setting up Libraries

We need to setup the default environment before we start.

- a) Click File -> Setup -> Defaults
- b) In the “Link library” field, put the following:  
**\* vst\_n18\_sc\_tsm\_c4\_wc.db tpz973gwc.db**
- c) In the “Target library” field, put the following:  
**vst\_n18\_sc\_tsm\_c4\_wc.db tpz973gwc.db**
- d) Make the “Symbol library” field blank.
- e) The window should now match the following figure. Click OK.

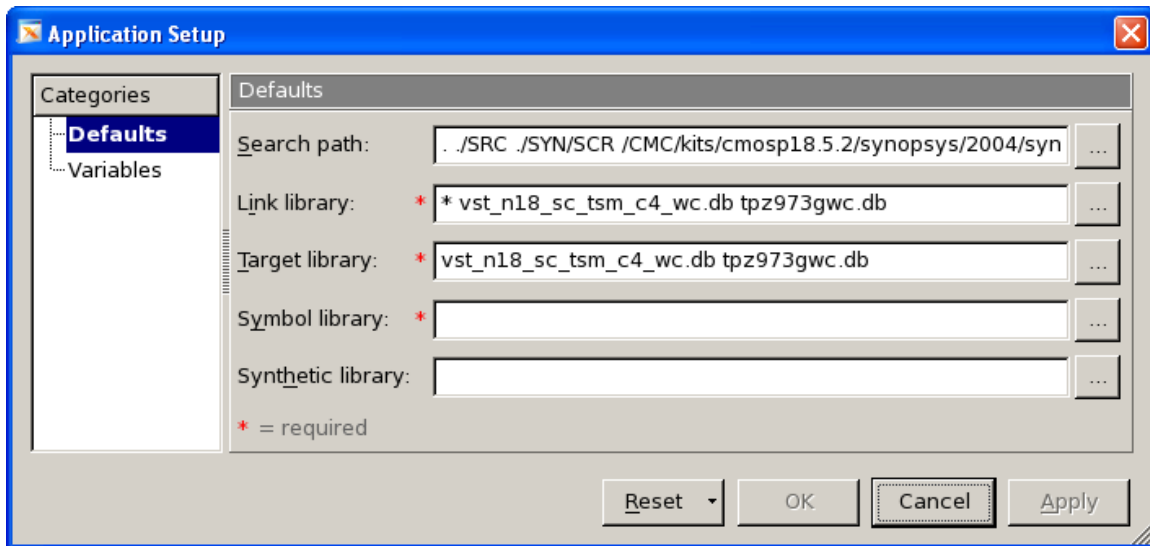


Figure 2.1

### 2.2 Reading the Design

There are 2 steps to read in a design: analyzing and elaborating.

a) Analyze the Design

The analysis command reads the HDL source and checks for syntactical errors. Read the source files using the following command:

```
$ analyze -library WORK -format vhd1 {./SRC/ha.vhd ./SRC/fa.vhd ./SRC/rca.vhd }
```

The same command can be performed by File -> Analyze...

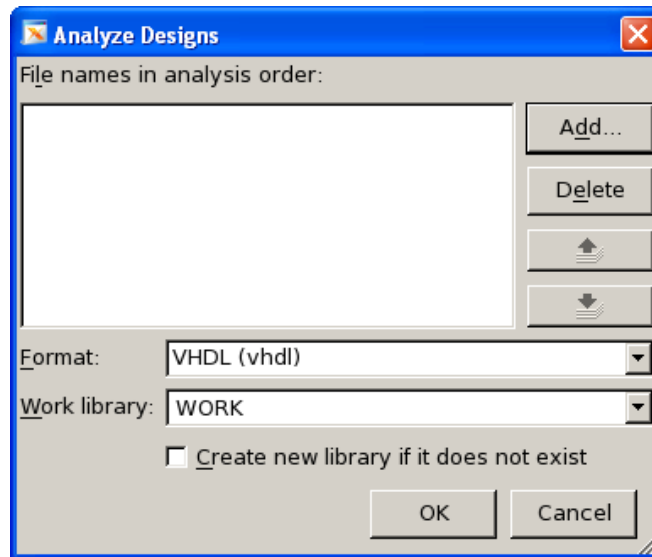


Figure 2.2

Select Add... and browse into SRC directory and select your vhd1 files. Check that Format is VHDL and Work library is WORK, and click OK.

b) Elaborate the Design

The next step is to elaborate the design. The design is translated into a technology-independent design (GTECH). Changing of parameter values defined in the source code is allowed in this step.

```
elaborate RCA -architecture STR -library WORK -parameters "N=4"
```

The same command can be found in File -> Elaborate...



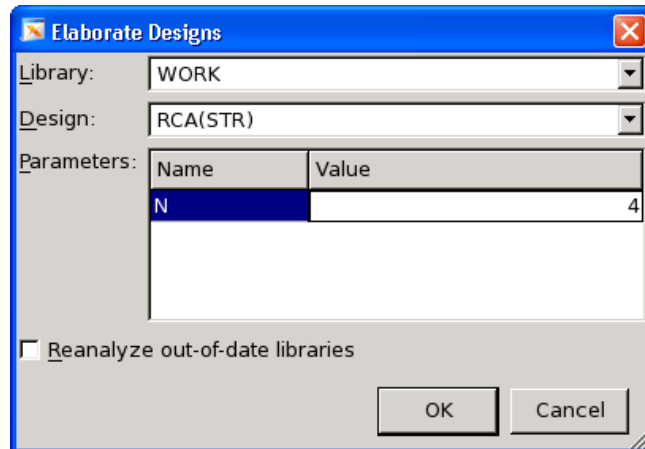


Figure 2.3


Make sure that Library is WORK, Design is RCA(STR), and Parameter N is set to 4.

You can see the following messages in the console about the elaboration process:

```
dc_shell> elaborate RCA -architecture STR -library WORK -parameters "N=4"
Running PRESTO HDLC
```

```
Presto compilation completed successfully.
Elaborated 1 design.
Current design is now 'rca_N4'.
Information: Building the design 'fa'. (HDL-193)
Presto compilation completed successfully.
Information: Building the design 'ha'. (HDL-193)
Presto compilation completed successfully.
```

We can also check the design structure in a more straightforward way:

The design schematic can be generated with the “Create Design Schematic” button . The following figures show the design’s structure after elaboration:

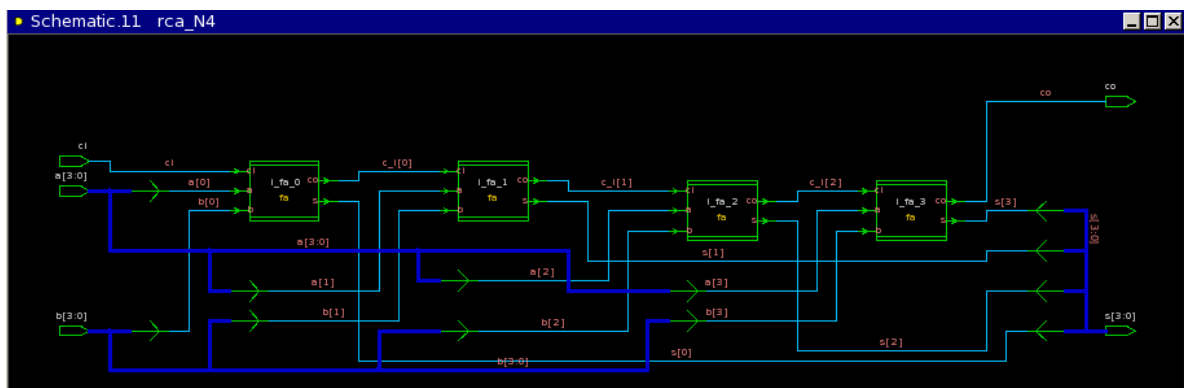


Figure 2.4

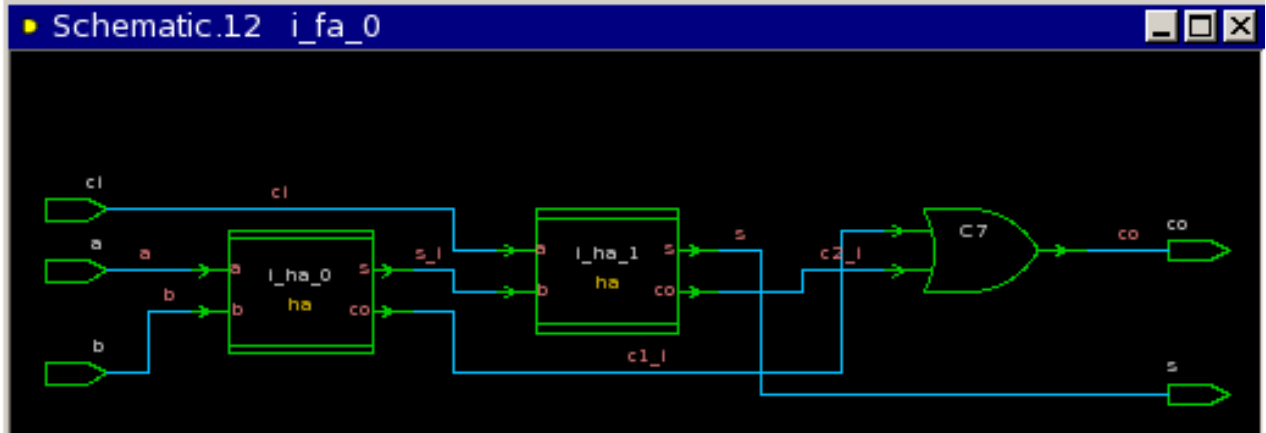


Figure 2.5

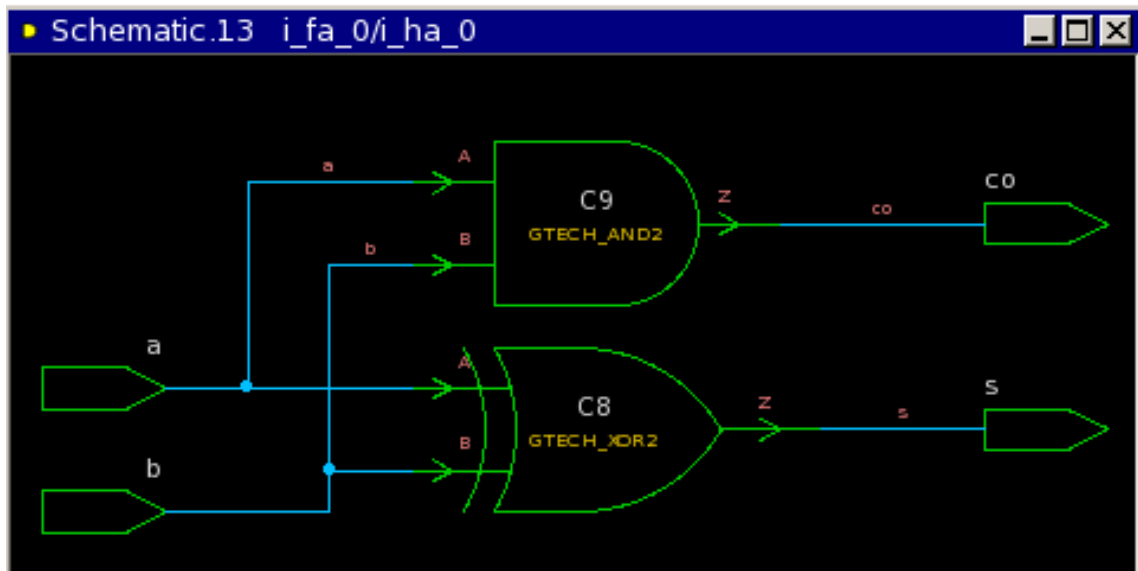


Figure 2.6

These figures represent the design structure as it was understood and extracted by Design Compiler from your HDL description. This design structure will be the starting point for the optimization that follows, but it doesn't necessarily represent the final structure.

Also note that the design is currently represented in **GTECH** cells, which are technology-independent formats internally used by Design Compiler. We will replace them with actual library cells later.

### c) Check the Design

Now the design is read into the Design Compiler memory, and it was analyzed for syntactical errors and non-synthesizable structures. However, there might still be some other problems with the design. We need to run the “check\_design” command to check the design consistency.

**\$ check\_design -multiple\_designs**

You will see the following message in the console:

```
dc_shell> check_design -multiple_designs
Information: Design 'fa' is instantiated 4 times. (LINT-45)
    Cell 'i_rca/i_fa_0' in design 'adder_N4'
    Cell 'i_rca/i_fa_1' in design 'adder_N4'
    Cell 'i_rca/i_fa_2' in design 'adder_N4'
    Cell 'i_rca/i_fa_3' in design 'adder_N4'
Information: Design 'ha' is instantiated 8 times. (LINT-45)
    Cell 'i_rca/i_fa_0/i_ha_0' in design 'adder_N4'
    Cell 'i_rca/i_fa_0/i_ha_1' in design 'adder_N4'
    Cell 'i_rca/i_fa_1/i_ha_0' in design 'adder_N4'
    Cell 'i_rca/i_fa_1/i_ha_1' in design 'adder_N4'
    Cell 'i_rca/i_fa_2/i_ha_0' in design 'adder_N4'
    Cell 'i_rca/i_fa_2/i_ha_1' in design 'adder_N4'
    Cell 'i_rca/i_fa_3/i_ha_0' in design 'adder_N4'
    Cell 'i_rca/i_fa_3/i_ha_1' in design 'adder_N4'
```

The message above gives us information about ‘fa’ and ‘ha’ being instantiated multiple times. That is how the adder is designed in our vhdl files, so we can ignore this warning.

### d) Save the Elaborated Design

It is recommended to save your design every now and then during the synthesis flow. Use “write\_file” command to write the elaborated design into a Design Compiler database:

**\$ write\_file -format ddc -hierarchy -output ./SYN/DDC/rca\_N4\_elab.ddc**

The same command from File -> Save As...

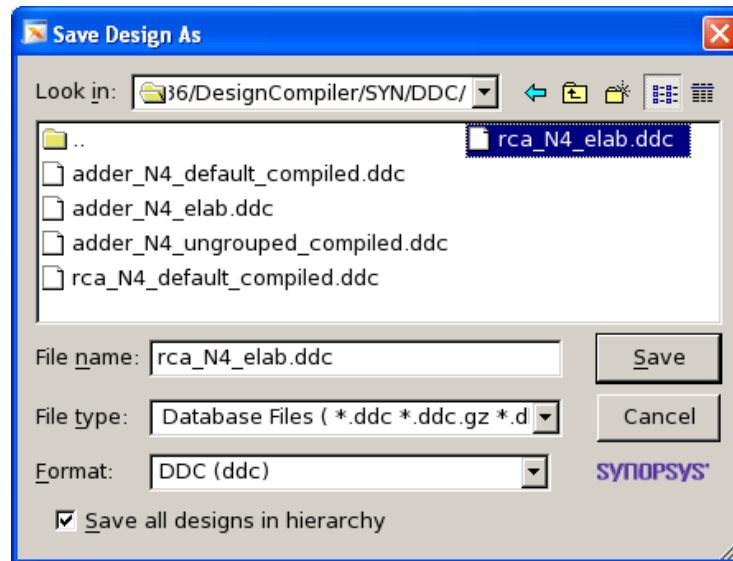


Figure 2.7

## 2.3 Setting Constraints

Next, we will set the design constraints and define the design environment. This is the most important step since the synthesis results depend on the constraints you set.

### a) System interface

First, we need to provide information about the logic that is driving the inputs and the capacitive loads at the outputs.

In this example, we assume the system has the interface shown below. Signals of input port A are driven by D-flipflops (DFFPQ1) and the signals from the output port S is captured at the next D-flipflop (DFFPQ1).

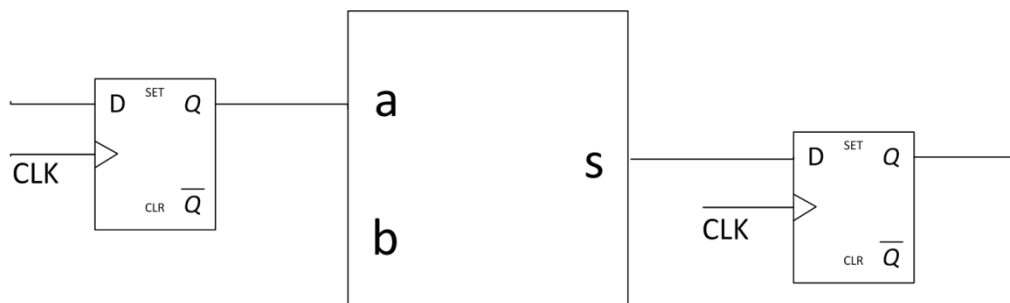


Figure 2.8

```
set_driving_cell -library vst_n18_sc_tsm_c4_wc -lib_cell DFFPQ1 -pin Q [get_ports a]
set_load [load_of vst_n18_sc_tsm_c4_wc/DFFPQ1/D] [get_ports s]
```

You can also click on each pin and go to Attributes -> Operating Environment -> Drive Strength

The following warnings will show up when you set the driving cells:

```
dc_shell> set_driving_cell -library vst_n18_sc_tsm_c4_wc -lib_cell DFFPQ1 -pin Q [get_ports a]
```

Warning: Design rule attributes from the driving cell will be set on the port. (UID-401)

Warning: Design rule attributes from the driving cell will be set on the port. (UID-401)

Warning: Design rule attributes from the driving cell will be set on the port. (UID-401)

Warning: Design rule attributes from the driving cell will be set on the port. (UID-401)

1

These are just to notify you that the design rule attributes attached to the driving cells will be considered when optimizing the design.

#### b) Area

The next design constrain we need to set is the maximum area for the design

**set\_max\_area 1000**

Or go to Attributes -> Optimization Constraints -> Design Constraints -> Max area

#### c) Operation Environment

Now we will define the environment in which our design expects to operate. Operation environment includes parameters such as operating conditions (variations in temperature, voltage and process) and wire load models (used to specify the effects of the wires on timing and area).

**set\_operating\_conditions -library vst\_n18\_sc\_tsm\_c4\_wc TYPICAL**

This can also be set by Attributes -> Operating Environment -> Operating Conditions..

**set\_wire\_load\_model -name TSMC128K\_Aggressive -library tpz973gwc**  
**set\_wire\_load\_mode top**

Or Attributes -> Operating Environment -> Wire Load...

## 2.4 Synthesizing the Design

Now that the design is constrained it is ready for synthesis. We will do some synthesis runs to illustrate different synthesis techniques and their effects on the results.

First, we will do a default synthesis run which does not apply any special optimizations and will be used as a reference for the other synthesis runs. Use the following command:

**compile**

The same command can be found from Design -> Compile Design:

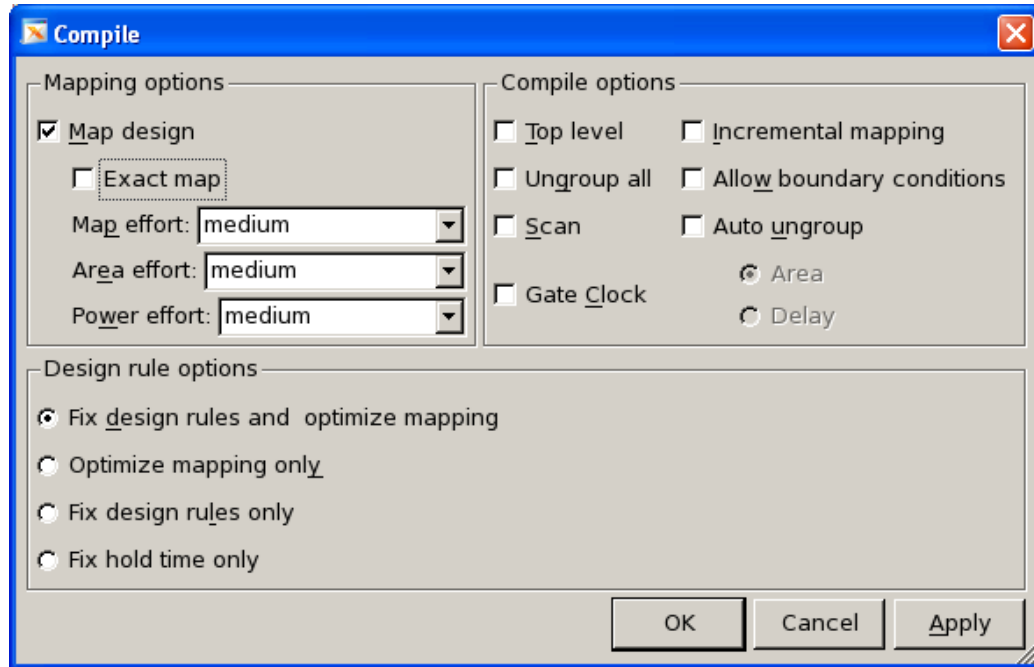


Figure 2.9

When the compilation is completed, you should see the following results by regenerating the schematics.

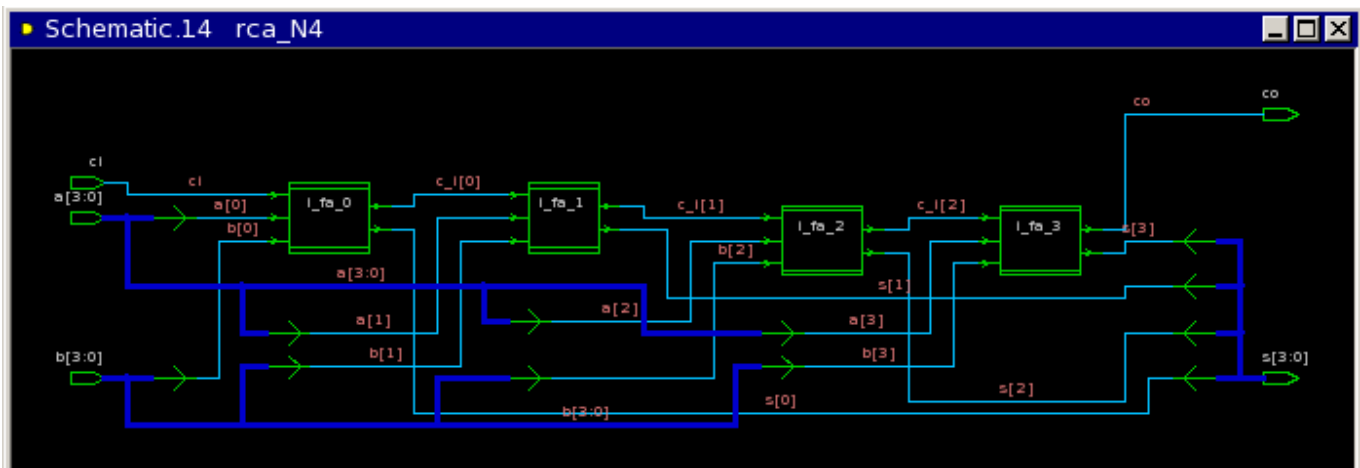


Figure 2.10

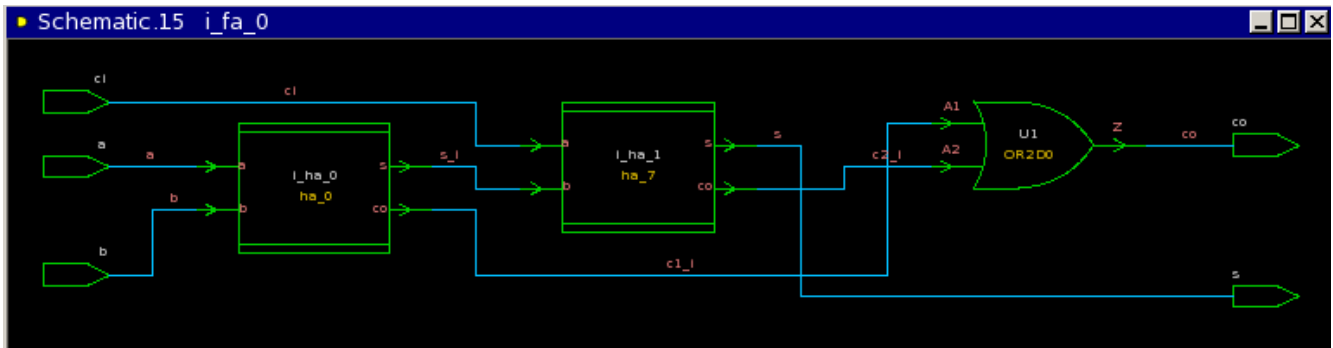


Figure 2.11

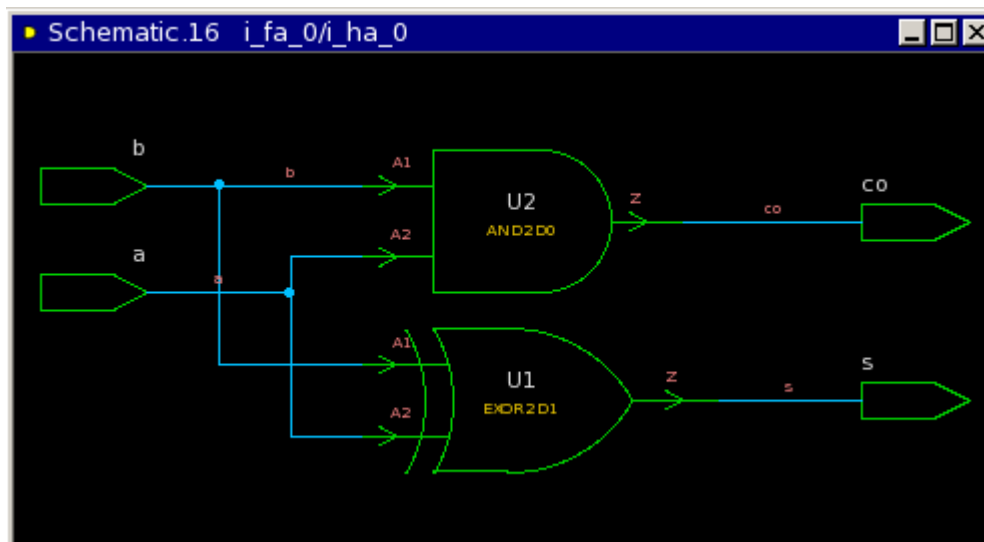


Figure 2.12

These schematics match quite accurately the structure of our HDL design description. Note in Figure 2.11 and 2.12, the design is now finally mapped to the actual technology library cells (OR2D0 / AND2D0 / EXOR2D1) instead of GTECH cells.

## 2.5 Design Analysis and Reporting

We will now look at how to analyze the synthesis results.

### a) Reporting Violations

To obtain a basic idea about our design, we need to get some general statistics like timing, design rule violations, area, structure, and so on from it. The “report\_qor” (QoR=Quality of Results) command provides lots of information which may pinpoint the possible design problems and what areas in our design need more detailed analyses. Note: We will save all the reports into the ./SYN/RPT directory so that they can be examined later.

```
report_qor > ./SYN/RPT/report_qor_default.txt
```

Use “report\_qor” only to display the report in the console:

```
dc_shell> report_qor
Information: Updating design information... (UID-85)
*****
Report : qor
Design : rca_N4
Version: D-2010.03-SP4
Date   : Tue Oct 1 11:08:38 2013
*****

Timing Path Group (none)
-----
Levels of Logic:          8.00
Critical Path Length:      2.02
Critical Path Slack:      uninit
Critical Path Clk Period:  Undef
Total Negative Slack:      0.00
No. of Violating Paths:    0.00
Worst Hold Violation:      0.00
Total Hold Violation:      0.00
No. of Hold Violations:    0.00
-----

Cell Count
-----
Hierarchical Cell Count:   12
Hierarchical Port Count:   52
Leaf Cell Count:        20
Buf/Inv Cell Count:        0
CT Buf/Inv Cell Count:     0
-----

Area
-----
Combinational Area:    455.344002
Noncombinational Area: 0.000000
Net Area:                  0.000000
-----
Cell Area:            455.344002
Design Area:               455.344002

Design Rules
-----
Total Number of Nets:      29
Nets With Violations:      0
```



-----  
 Hostname: bimos

### Compile CPU Statistics

-----  
 Resource Sharing: 0.00  
 Logic Optimization: 0.04  
 Mapping Optimization: 0.67  
 -----  
 Overall Compile Time: 1.67

The above report does not indicate any major problems in our design. Since there is no sequential logic in our design, we will not have any hold time violation.

### b) Timing Analysis

The next thing we can check is timing. The “report\_timing” command provides timing-related info of the current design. By default, it displays info on the critical path or the path with maximum delay.

**report\_timing > ./SYN/RPT/report\_timing\_default.txt**

“report\_timing” will give you the following report:

```
*****
Report : timing
-path full
-delay max
-max_paths 1
Design : rca_N4
Version: D-2010.03-SP4
Date   : Tue Oct 1 11:39:34 2013
*****

Operating Conditions: TYPICAL  Library: vst_n18_sc_tsm_c4_wc
Wire Load Model Mode: top

Startpoint: a[0] (input port)
Endpoint: s[3] (output port)
Path Group: (none)
Path Type: max

Des/Clust/Port  Wire Load Model      Library
-----
rca_N4         TSMC128K_Aggressive        tpz973gwc
```

Point	Incr	Path
input external delay	0.00	0.00 r
a[0] (in)	0.04	0.04 r
i_fa_0/a (fa_0)	0.00	0.04 r
i_fa_0/i_ha_0/a (ha_0)	0.00	0.04 r
i_fa_0/i_ha_0/U1/Z (EXOR2D1)	0.46	0.50 f
i_fa_0/i_ha_0/s (ha_0)	0.00	0.50 f
i_fa_0/i_ha_1/b (ha_7)	0.00	0.50 f
i_fa_0/i_ha_1/U2/Z (AND2D0)	0.15	0.65 f
i_fa_0/i_ha_1/co (ha_7)	0.00	0.65 f
i_fa_0/U1/Z (OR2D0)	0.21	0.86 f
i_fa_0/co (fa_0)	0.00	0.86 f
i_fa_1/ci (fa_3)	0.00	0.86 f
i_fa_1/i_ha_1/a (ha_5)	0.00	0.86 f
i_fa_1/i_ha_1/U2/Z (AND2D0)	0.15	1.01 f
i_fa_1/i_ha_1/co (ha_5)	0.00	1.01 f
i_fa_1/U1/Z (OR2D0)	0.21	1.22 f
i_fa_1/co (fa_3)	0.00	1.22 f
i_fa_2/ci (fa_2)	0.00	1.22 f
i_fa_2/i_ha_1/a (ha_3)	0.00	1.22 f
i_fa_2/i_ha_1/U2/Z (AND2D0)	0.15	1.38 f
i_fa_2/i_ha_1/co (ha_3)	0.00	1.38 f
i_fa_2/U1/Z (OR2D0)	0.21	1.58 f
i_fa_2/co (fa_2)	0.00	1.58 f
i_fa_3/ci (fa_1)	0.00	1.58 f
i_fa_3/i_ha_1/a (ha_1)	0.00	1.58 f
i_fa_3/i_ha_1/U1/Z (EXOR2D1)	0.43	2.02 f
i_fa_3/i_ha_1/s (ha_1)	0.00	2.02 f
i_fa_3/s (fa_1)	0.00	2.02 f
s[3] (out)	0.00	2.02 f
data arrival time		2.02

(Path is unconstrained)

The data arrival time is the time required for signal to travel from path start point to a path end point. From the data arrival calculation we can see the critical path starts from the input pin **a[0]** and ends at the output pin **s[3]**. In between are shown all the cells on the path. Examining this path reveals that the critical path goes through all full adders in the design which is exactly what we can expect from a Ripple Carry Adder.

We can examine the full critical path by selecting Schematic -> New Path Schematic View -> Of Paths From/Through/To...

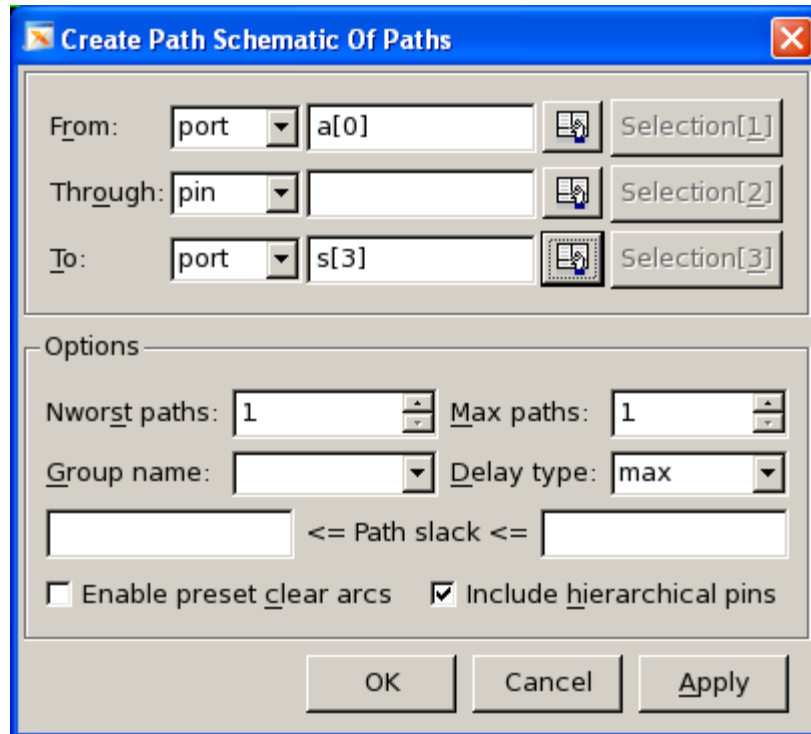


Figure 2.13

You should see the path below which shows the critical path from path start point (a[0]) to path end point (s[3]) in a single view.

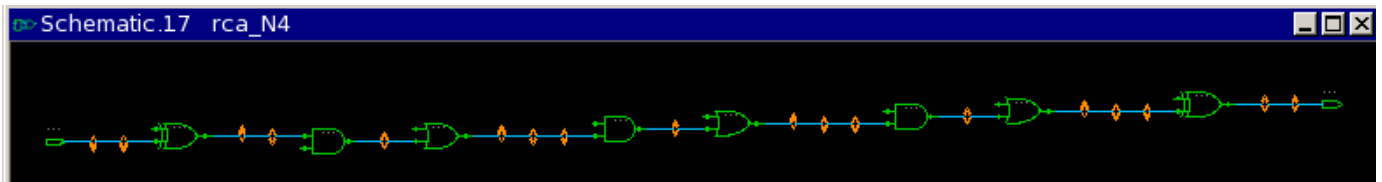


Figure 2.14

### c) Structure Analysis

The area of a design is one of the primary interests of synthesis results. The “report\_area” command displays info on the design’s area.

```
report_area > ./SYN/RPT/report_area_default.txt
```

This produces the following report:

```
*****
Report : area
Design : adder_N4
Version: D-2010.03-SP4
```

Date : Tue Oct 1 12:46:05 2013

\*\*\*\*\*

Library(s) Used:

vst\_n18\_sc\_tsm\_c4\_wc (File: /CMC/kits/cmosp18.5.2/synopsys/2004/syn/vst\_n18\_sc\_tsm\_c4\_wc.db)

Number of ports: 14  
 Number of nets: 17  
 Number of cells: 4  
 Number of references: 4  
  
 Combinational area: 455.344002  
 Noncombinational area: 0.000000  
 Net Interconnect area: undefined (Wire load has zero net area)  
  
 Total cell area: 455.344002  
 Total area: undefined

The above report displays the area information of the design. The total area is the sum of three factors: Combinational, Noncombinational, and Net Interconnect area. The area due to logic cells in design is shown by the combinational (basic logic gates like ANDs, ORs, and the like) and the noncombinational (registers) factors. There is no register in our design, so the noncombinational area is 0. The third factor affecting the area (net interconnect area) is due to the wires connecting these cells. In this library the wire load models do not include area information (Wire load has zero net area) so the Net Interconnect area (and therefore Total area) is left undefined.

#### d) Synthesis results summary

	Default run
Combinational Area / $\mu\text{m}^2$	455.344002
Noncombinational Area / $\mu\text{m}^2$	0.000000
Total Cell Area / $\mu\text{m}^2$	455.344002
Data arrival time /ns	2.02
Levels of Logic	8
Leaf Cell Count	20

## 2.6 Saving the Design

Design Compiler will not automatically save your design when exiting, so we need to save necessary files from the synthesis. There are a few different formats that you can save your design into. For this tutorial, you should save the design in the following 2 formats: .ddc (compiled design database) and .v (Verilog netlist of your design). The database is needed if you

need to come back and fix something in your design later on. The Verilog netlist is needed as an input for the back-end flow that follows (e.g. Placement and Routing).

```
write_file -format ddc -hierarchy -output ./SYN/DDC/rca_N4_default_compiled.ddc  
write_file -format verilog -hierarchy -output ./SYN/NETLIST/rca_N4_default_compiled.v
```

Or go to File -> Save As, and select Format: DDC (ddc) or VERILOG (v).

## 2.7 Optimizations

### a) Ungrouping Full Adders

The example adder HDL description is not perfectly written for synthesis. When optimizing the design, Design Compiler operates only within a single block at a time. This means that if your design has very small partitions, they actually prevent Design Compiler from performing optimizations on your design. Therefore this design has a couple of design flaws from the synthesis point of view:

- There are too many levels of hierarchy (ha->fa->rca)
- Hierarchical cells are too small (ha)

So, to better synthesize the design, we need to remove (ungroup) the hierarchy inside all the Full Adder cells `i_fa_{0|1|2|3}` with the following command:

```
ungroup [ get_cells i_fa_*/* -filter "is_hierarchical == true" ]
```

Now the `i_fa_{0|1|2|3}` cells should look like this:

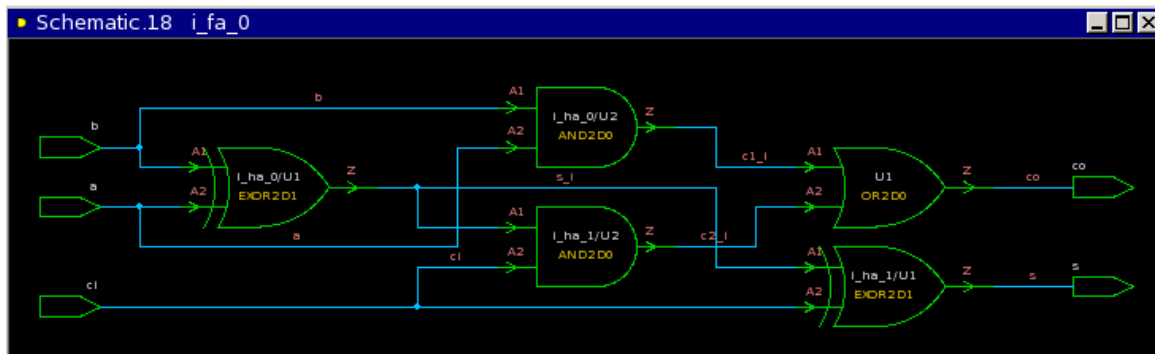


Figure 2.15

### b) Optimization

Now we have modified the design structure, to re-optimize the design, we just recompile it:

```
compile
```

Re-create the schematics, and we can see significant changes to the ungrouped Full Adder. An OAI (OR-AND-INV) cell has replaced the original 2 AND gates and 1 OR gate.

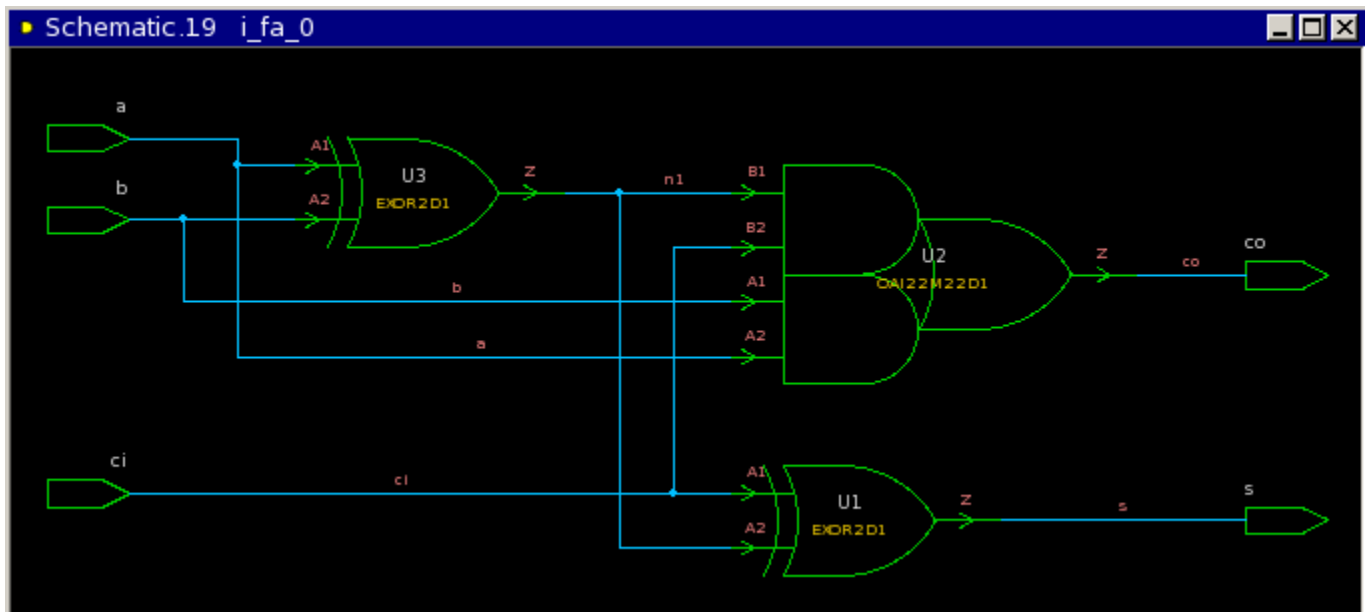


Figure 2.16

Check the “report\_qor” of your optimized design; you should see fewer levels of logic, improved timing, and less area:

	Default run	Ungrouped
Combinational Area /um <sup>2</sup>	455.344002	374.036011
Noncombinational Area /um <sup>2</sup>	0.000000	0.000000
Total Cell Area /um <sup>2</sup>	455.344002	374.036011
Critical Path Length /ns	2.02	1.63
Levels of Logic	8	5
Leaf Cell Count	20	12

c) Save your optimized design with name “rca\_N4\_ungrouped\_compiled” in .ddc and .v formats. **We will need the .v file for Lab 6.**

d) If you're ready for the Exercise, type “remove\_design -all” to reset Design Compiler.

Now we have finished the Design Compiler synthesis flow.

**Exercise:**

In this exercise, you are asked to synthesize a 4 bit comparator. The Verilog file *compar.v* describing the comparator is already provided. Download it from the course website, and save it in ./SRC as well.

- Take a look at the Verilog file, try to figure out what it does
- Synthesize the comparator in the following steps:
  - Analyze and elaborate design
  - Set constraints (apply the same driving cell at input a and output load, area constraint and operating environment)
  - Compile design (default run)
  - Report QoR, Timing and Area (Remember to change the report names accordingly before saving them so they won't overwrite you old reports)
  - Save your design in .ddc and .v formats. **You will need the .v file for Lab 6 exercise.**

**Report:**

1. Paste all the commands you used (in the right order). You can find them in the "History" tab above the console command line.
2. Schematics: (1) The comparator design before and after compilation;  
(2) The critical path.
3. The QoR report of your design.
4. Calculate the maximum frequency in which the comparator can work. (Hint: use critical path delay as the maximum propagation delay for combinational logic)  
Make sure D Flip-flop (DFFPQ1) is used as both driving cell and output load of the comparator. DFFPQ1 has the following parameters:

	Setup Time ( $t_{\text{setup}}$ )	clk to Q propagation delay ( $t_{\text{pcq}}$ )
DFFPQ1	0.085 ns	0.171 ns