SYNOPSYS®

CUSTOMER EDUCATION SERVICES

Design Compiler 1 Workshop

Student Guide

10-I-011-SSG-013 2007.03

Synopsys Customer Education Services
700 East Middlefield Road
Mountain View, California 94043

Workshop Registration: **1-800-793-3448**

www.synopsys.com

Document Order Number: 10-I-011-SSG-013
Design Compiler 1 Student Guide

Table of Contents

Unit 1: Introduction to Synthesis

Unit 2: Setting Up and Saving Designs

Table of Contents

Table of Contents

Unit 3: Design and Library Objects

Table of Contents

Unit 4: Area and Timing Constraints

Table of Contents

Table of Contents

Unit 6: Environmental Attributes

Table of Contents

Unit 7: Compile Commands

Table of Contents

Table of Contents

Table of Contents

Unit 8: Timing Analysis

Unit 9: More Constraint Considerations

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Unit 12: Pre- and Post-Synthesis Considerations

Table of Contents

Unit 13: Conclusion

Design Compiler 1

2007.03

Synopsys Customer Education Services

Synopsys 10-I-011-SSG-013

Introductions

- Name
- Company
- Job Responsibilities
- EDA Experience
- Main Goal(s) and Expectations for this Course

1- 2

EDA = Electronic Design Automation

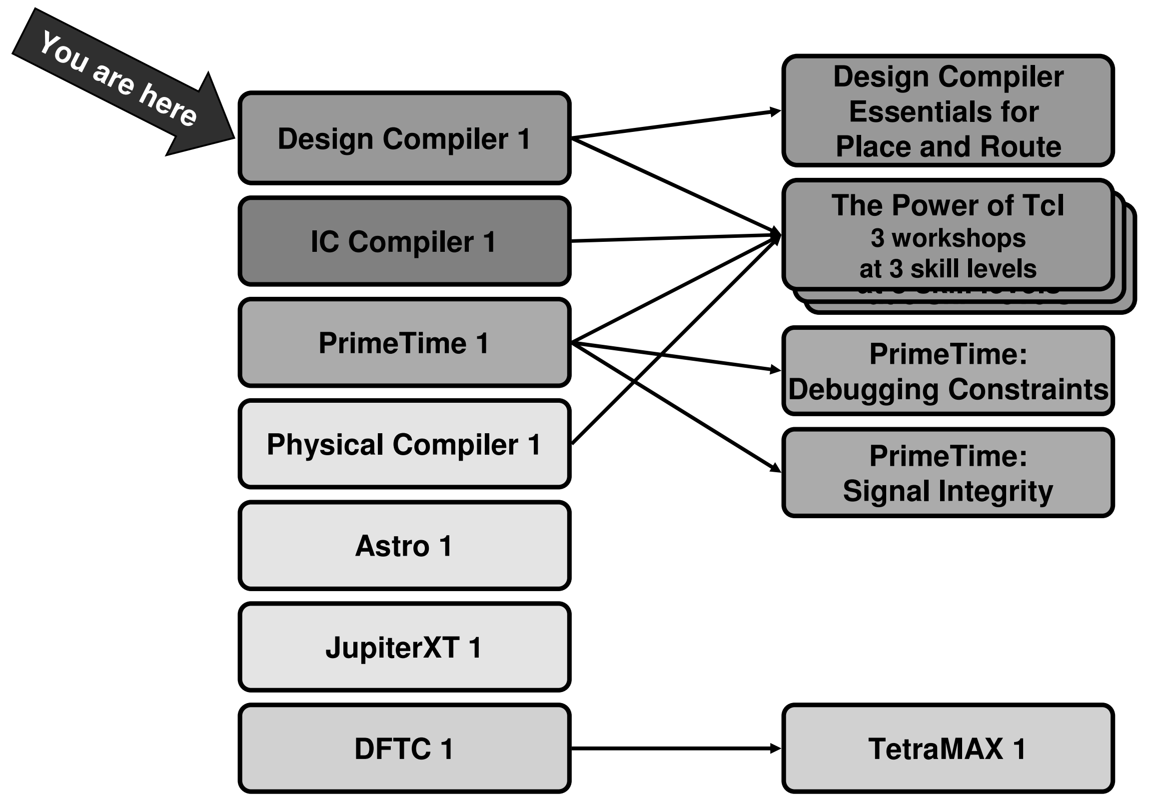
Facilities

Building Hours	Phones
Emergency EXIT	Messages
Restrooms	Smoking
Meals	Recycling

Please turn off cell phones and pagers

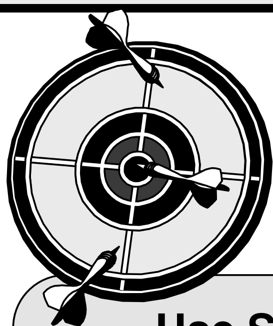
1-3

Curriculum Flow



Synopsys Customer Education Services offers workshops in two formats: The “classic” workshops, delivered at one of our centers, and the virtual classes, that are offered conveniently over the web. Both flavors are delivered *live* by expert Synopsys instructors.

Workshop Goal



Use Synopsys' Design Compiler to:

- Constrain a complex design for area and timing
- Apply synthesis techniques to achieve area and timing closure
- Analyze the results
- Generate design data that works with physical design or layout tools

1- 5

Target Audience

ASIC digital designers with little or no Design Compiler experience.

1- 6

Workshop Prerequisites

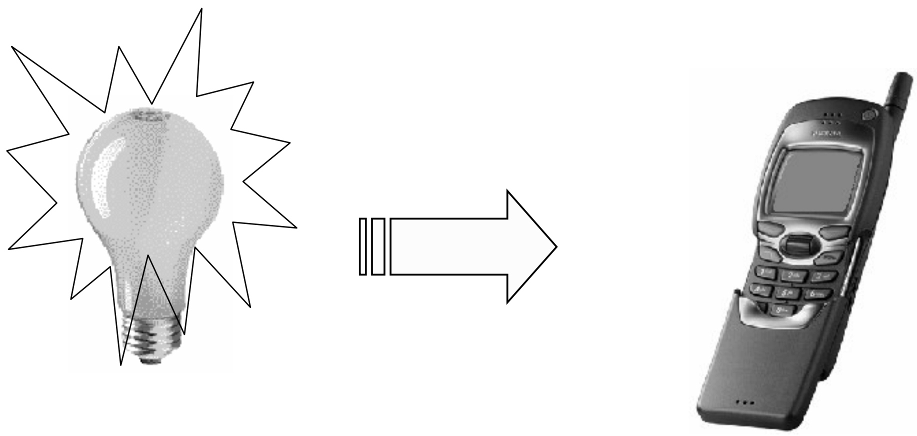
You should have knowledge of the following:

- Digital Logic
- UNIX and X-Windows
- A Unix based text editor

1- 7

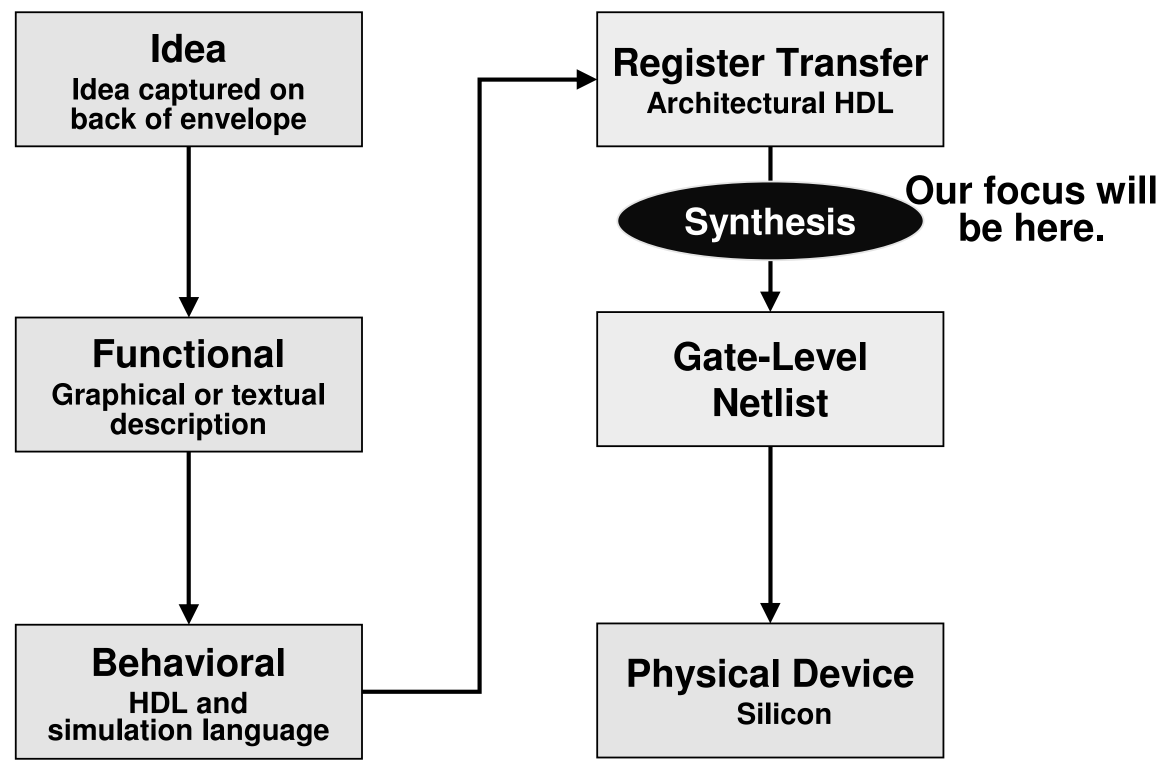
What Does “Synthesis” Mean?

Synthesis is the *transformation* of an idea into a manufacturable device to carry out an intended function.



1- 8

What do *WE* Mean by "Synthesis"?



1-9

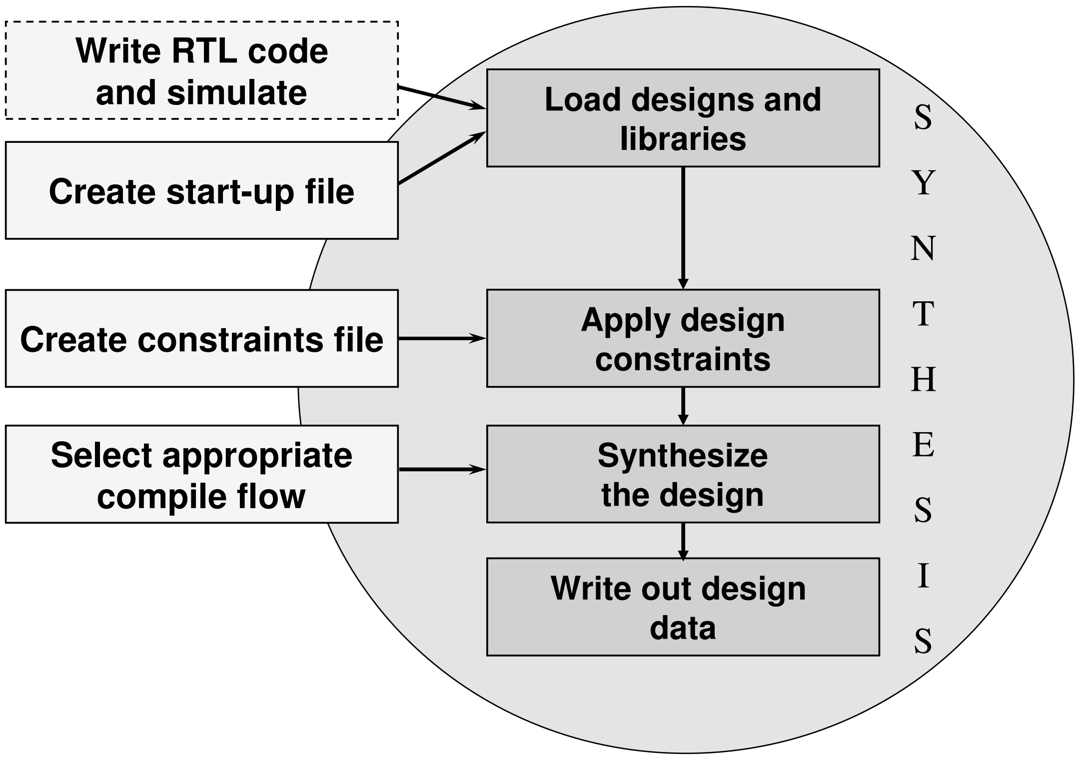
The **functional** description serves to validate the intent of your new design. It might use stochastic processes to verify the performance.

The **behavioral** description uses HDL code to model the design at system-level. It allows you to capture key algorithms and alternative architectures.

The **register transfer level** completely models your design in detail. All clocks are defined and all registers declared. Use this HDL model as input for Design Compiler. You have enough details to synthesize trade-offs in area and timing.

The input for 'Place & Route' tools is a **gate-level** netlist. The physical tools generate a GDSII file for the chip manufacturing.

Design Compiler Flow



1- 10

This course covers the entire flow shown above, except for writing and simulating RTL code.

Synthesis Transformations

Synthesis = Translation + Logic Optimization + Gate Mapping

RTL Source

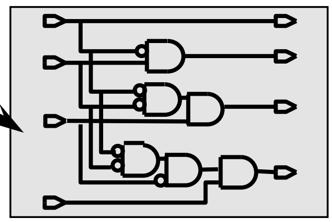
```
residue = 16'h0000;
if (high_bits == 2'b10)
   residue = state_table[index];
else
   state_table[index] = 16'h0000;
```

1 **Translate (`read_verilog` `read_vhdl`)**

Constraints

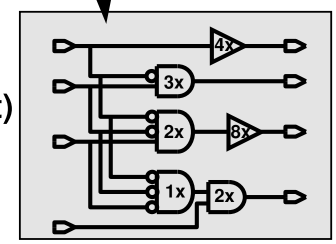
```
set_max_area …
create_clock ….
set_input_delay …
```

2 **Constrain (`source`)**



Generic Boolean Gates
(GTECH or unmapped *ddc* format)

3 **Optimize + Map**
(`compile`)



Technology-specific Gates
(mapped *ddc* format)

The verb "to compile" is used synonymously with "to synthesize"

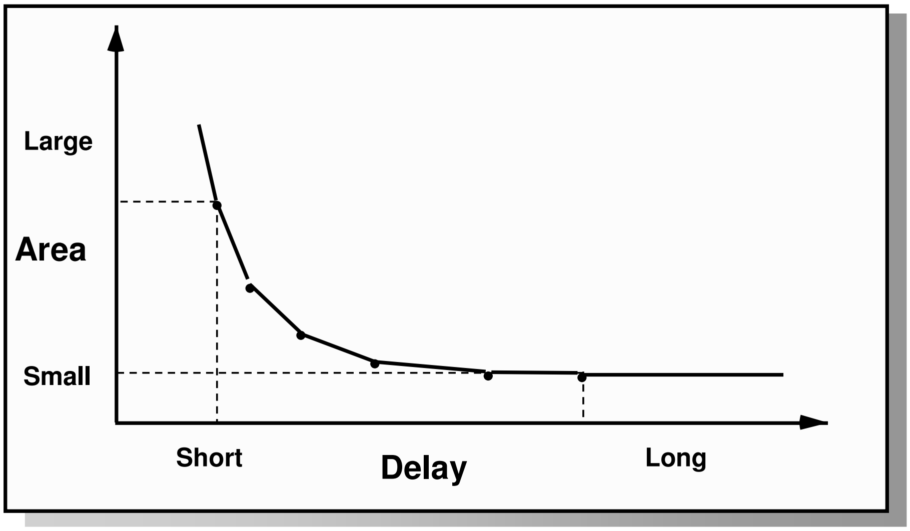
4 **Save (`write -f ddc`)**

1- 11

GTECH components have no timing or load characteristics, and are only used by Design Compiler.
ddc is, by default, an internal DC format, which can also explicitly be written out.

Note: *db* is the old DC format which has been replaced by *ddc* (in *XG mode*)

Synthesis Is Constraint-Driven



You set the goals (through constraints).

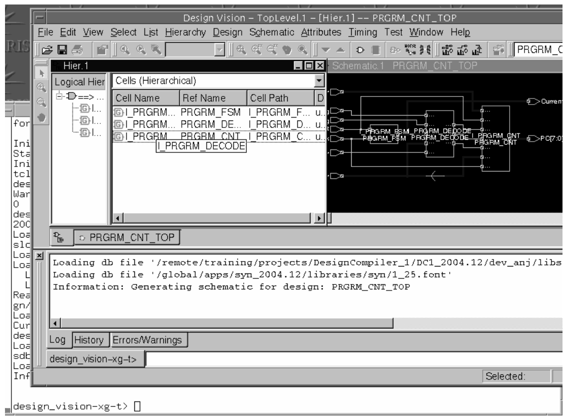
Design Compiler optimizes the design to meet your goals.

1- 12

The designer guides the synthesis tool by providing **constraints**, i.e. information about the timing and area requirements for the design. The synthesis tool uses this information and tries to generate the smallest possible design that will satisfy the timing requirements. Without any constraints specified, the synthesis tool will generate a non-optimal netlist, which might not satisfy the designer's requirements.

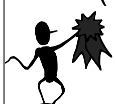
Three Interfaces to Design Compiler

1. Design Vision (interactive GUI) → `unix% design_vision`



2. DC Shell (interactive shell) → `unix% dc_shell-t`

```
unix% dc_shell-t
dc_shell-xg-t>
```



3. Batch mode → `unix% dc_shell-t -f RUN.tcl | tee -i my.log`

DC invoked in 'TCL mode' and 'XG mode'

1-13

As a beginner with Design Compiler the GUI interface provides a comfortable environment within which to learn the basic commands and flows in DC. Most commonly used commands are listed in pull-down menus so you do not need to memorize commands. There is also a DC-shell window at the bottom of the GUI in which the menu's underlying dc_shell commands are echoed, so you can familiarize yourself with the command syntax, and you can also type in the commands there. Once you are comfortable with the commands and analyzing reports without schematics, you can migrate to using the interactive DC-Shell environment. As you become very comfortable with DC you will want to create compile (or run) scripts along with constraints file(s), and you'll be able to take advantage of executing your synthesis jobs in batch mode.

The "-t" extension invokes DC shell in Tcl-mode, which is the recommended mode, and by default, also XG mode, which is also recommended. Design Vision is ONLY available in TCL mode, hence no "-t" extension is needed, and it also comes up in XG mode by default.

Tcl-mode requires that the commands used are in "DC-Tcl" syntax, as opposed to the older "dcsh" syntax. All commands in this workshop are in DC-Tcl.

For DC versions 2004-12 and 2005.09 it is necessary to explicitly include "-xg" to invoke DC in XG mode: `dc_shell`**`-xg`**`-t`, or `design_vision`**`-xg`**. (XG mode not available prior to 2004.12)

Design Vision replaces the much older Design Analyzer GUI, which is invoked with **`design_analyzer`**.

What is XG Mode (versus DB Mode)?

- **XG mode uses optimized memory management techniques that increase the tool's capacity and can reduce runtime**
- **The following Synopsys synthesis tools support the new XG mode**
  - Design Compiler
  - DFT Compiler
  - Power Compiler
  - Physical Compiler
- **In XG mode, all synthesis tools use the tool command language (Tcl)**
  - XG mode does not support the *dcsh* command language

1- 14

DB versus *XG* are two different memory management modes. *DB mode* is the default mode for synthesis tool version W-2005.09 and earlier. In general, *dc_shell* behaves the same in *DB mode* and *XG mode*, but *XG mode* can provide you with reduced memory consumption and runtime.

dcsh is a command language specific to Synopsys.

XG mode does not require any special licensing.

XG mode supports all existing hardware platforms, with the exception of HP32. (The Milkyway product does not support the HP32 platform.)

What Changes in XG Mode?

- **Use the new binary *.ddc* format to save design netlists**
  - Use in the same way as the old *.db* format

```
dc_shell-xg-t> read_ddc MYDES.ddc
dc_shell-xg-t> write -format ddc -hierarchy -output MYDES.ddc
```

- **Convert old *.db* to *.ddc* for maximum benefit**
  - Use of the *.db* format for storing designs is still possible, but highly discouraged - results in significant memory overhead

```
UNIX% dc_shell-t; # Invoke DC in XG mode
dc_shell-xg-t> read_db MYDES.db; # Reverts DC to 'DB mode'
dc_shell-xg-t> write -format ddc -hierarchy -output MYDES.ddc
dc_shell-xg-t> exit
UNIX% dc_shell-t; # re-invoke DC in XG mode and read MYDES.ddc
```

1-15

Formality supports the *.ddc* format beginning with the V-2004.06 release.

Physical Compiler XG mode, *PrimeTime*, and *PrimePower* support the *.ddc* format beginning with the W-2004.12 release

Helpful *UNIX-like* DC_Shell commands

Find the location and/or names of files[1]
```
dc_shell-xg-t> pwd; cd; ls
```

Show the history of commands entered:
```
dc_shell-xg-t> history
```

Repeat last command:
```
dc_shell-xg-t> !!
```

Execute command no. 7 from the history list:
```
dc_shell-xg-t> !7
```

Execute the last `report` command:
```
dc_shell-xg-t> !rep
```

Execute any UNIX command:
```
dc_shell-xg-t> sh <UNIX_command>
```

Get any UNIX variable value:
```
dc_shell-xg-t> get_unix_variable <UNIX_env_variable>
```
[2]

1-16

[1] Use `cd` very carefully, because you will change the relative starting point (**.**) for your directory `search_path`. By default "**.**", the current working directory, is set to the directory in which you invoked Design Compiler (shell or GUI).

[2] For example, use the following to determine if you are in a Sun or Linux environment:
`dc_shell-xg-t> get_unix_variable ARCH` → may return "*linux*" or "*sparcOS5*"

Agenda

DAY 1

1	Introduction to Synthesis
2	Setting Up and Saving Designs
3	Design and Library Objects
4	Area and Timing Constraints

1- 17

Agenda

DAY 2

5	Partitioning for Synthesis
6	Environmental Attributes
7	Compile Commands
8	Timing Analysis
9	More Constraint Considerations

1-18

Agenda

DAY 3

9	More Constraint Considerations (Lab cont'd)
10	Multiple Clock/Cycle Designs
11	Synthesis Techniques and Flows
12	Post-Synthesis Output Data
13	Conclusion

1- 19

Summary: Exercise

Synthesis = ___________ + ______________ + ___________

If you are given a *.ddc* file you know that this design has been synthesized or compiled.

True or False?

1- 20

Translation + Logic Optimization + Gate Mapping

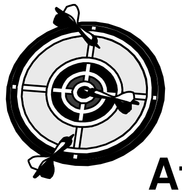
False.

Agenda

DAY 1

1	Introduction to Synthesis	
2	Setting Up and Saving Designs	
3	Design and Library Objects	
4	Area and Timing Constraints	

Unit Objectives

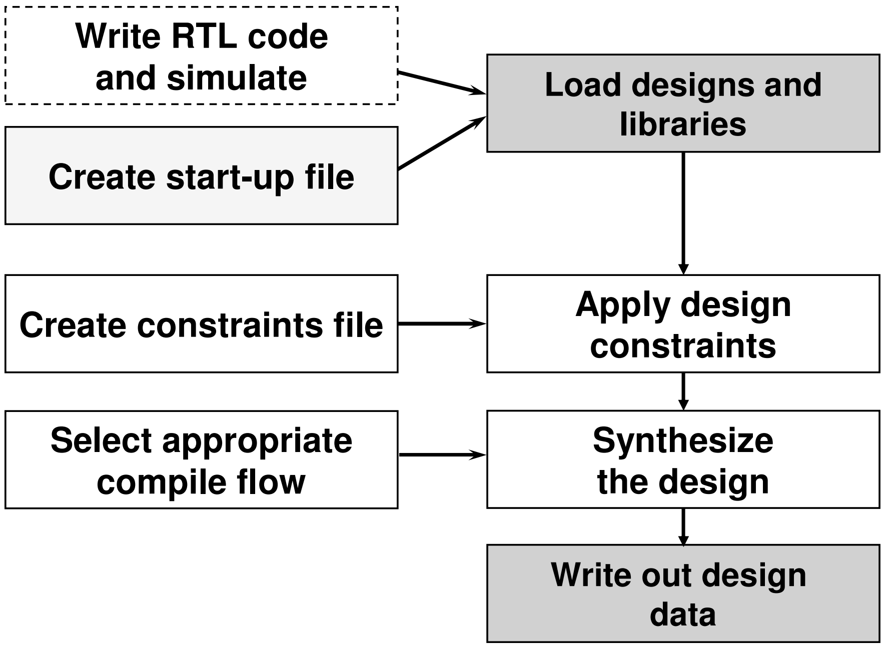


After completing this unit, you should be able to prepare a design for compile:

- **Create a DC *setup file* to specify the technology *library file* and *search path* directories**
- **Read in hierarchical designs**
- **Apply a constraints file**
- **Save the design**

2- 2

RTL Synthesis Flow



2- 3

This course covers the entire flow shown above, except for writing and simulating RTL code.

Commands Covered in this Unit

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog1 {A.v B.v TOP.v}
dc_shell-xg-t> current_design TOP
dc_shell-xg-t> link
dc_shell-xg-t> check_design
dc_shell-xg-t> write -f ddc -hier -out unmpd/TOP.ddc
dc_shell-xg-t> source -echo -verbose TOP.con
dc_shell-xg-t> check_timing
dc_shell-xg-t> compile -boundary -scan -map high
dc_shell-xg-t> report_constraint -all_violators
dc_shell-xg-t> change_names -rule verilog -hier
dc_shell-xg-t> write -f verilog -hier -out mpd/TOP.v
dc_shell-xg-t> write -f ddc -hier -out mpd/TOP.ddc
dc_shell-xg-t> exit
```

Invoke DC from the project directory (CWD)

"Good practice" steps

2- 4

[1] Other "read" commands will also be shown – the latter two are shown in the Appendix:

```
read_vhdl
read_ddc
analyze + elaborate
acs_read_hdl
```

Setup Commands/Variables in this Unit

.synopsys_dc.setup

```
set search_path "$search_path mapped rtl libs cons"
set target_library 65nm.db
set link_library "* $target_library"
set symbol_library 65nm.sdb

history keep 200
alias h history
alias rc "report_constraint -all_violators"
```

2- 5

Unit Agenda

Loading Designs and Libraries

DC Startup File

Loading Hierarchical Designs

2- 6

Invoking DC and Reading a Verilog RTL File

CWD: The directory that DC is invoked from, a.k.a. "project directory"

```
risc_design (CWD)
    |
  rtl/
  └ MYREG_rtl.v
```

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog rtl/MYREG_rtl.v

Loading db file `../libraries/syn/gtech.db'
Loading db file `../libraries/syn/standard.sldb'
Loading verilog file `../risc_design/MYREG_rtl.v'
 ...
Warning: Can't read link_library file `your_library.db'.
 ...
Current design is `MYREG'
```

Will be handled shortly

A design called *MYREG* is loaded in DC memory

The *read* command:

- **Loads default and specified libraries**
- **Reads RTL file(s) and translates into GTECH**
- **Loads the unmapped *ddc* design in DC memory**
- **Sets the *current design***

2- 7

CWD: Current Working Directory. To read a VHDL file use the command: *read_vhdl <file_name>*.

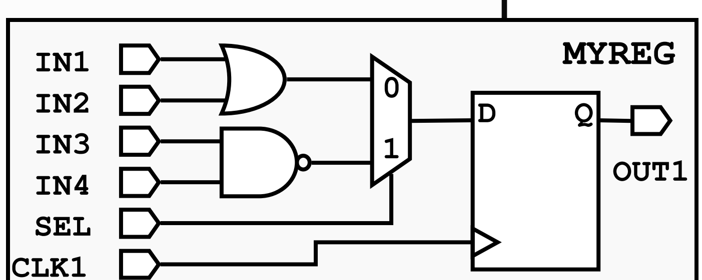
`dc_shell-t` invokes DC in TCL mode. Starting with v2006.06 XG mode is automatically selected.

gtech.db and *standard.sldb* are Synopsys-provided default libraries containing basic GTECH logic elements and basic DesignWare IP blocks, respectively. These libraries are automatically loaded by the *read* command. The read command also loads the library(ies) specified by the *link_library* variable, which, by default, is set to a non-existent file *your_library.db*, hence the Warning.

Example RTL file:

MYREG_rtl.v

```
module MYREG (IN1, IN2, IN3, IN4, SEL, CLK1, OUT1);
   input IN1, IN2, IN3, IN4, SEL, CLK1;
   output OUT1;
   reg OUT1;
       always @ (posedge CLK1)
       begin
        if (SEL)
          OUT1 <= ~(IN3 & IN4);
        else
          OUT1 <= IN1 | IN2;
       end
endmodule
```



Reading a VHDL RTL File

```
                                                  risc_design (CWD)
                                                        |
 ARCH/  ENTI/  MYREG-RTL.syn  MYREG.mr   MYREG.syn   rtl/
                                                     └ MYREG_rtl.vhd
```

UNIX% **cd risc_design**

UNIX% **dc_shell-t**

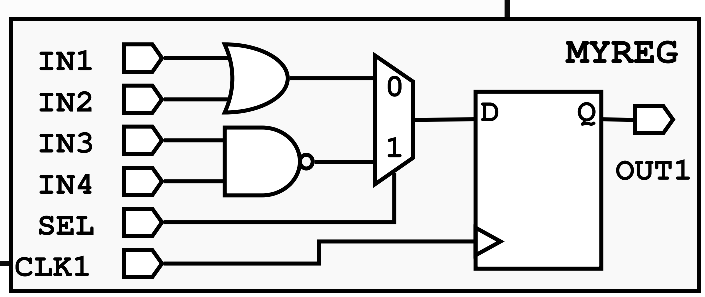
dc_shell-xg-t> **read_vhdl rtl/MYREG_rtl.vhd**

The *read_vhdl* command creates several intermediate files and directories which collectively form the "VHDL Design Library". Unless specified otherwise in your code, the VHDL design library is called 'WORK', by default. This is a VHDL-only concept. The 'WORK' files and directories are placed, by default, in the CWD.

2-8

MYREG_rtl.vhd

```
entity MYREG is port (IN1, IN2, IN3, IN4, SEL, CLK1: in STD_LOGIC;
                      OUT1: out STD_LOGIC);
end MYREG;

architecture RTL of MYREG is
begin
  process
   begin
   wait until CLK1'event and CLK1 = '1';
       begin
        if (SEL = '1') then
          OUT1 <= IN3 nand IN4;
        else
          OUT1 <= IN1 or IN2;
       end if;
  end process;
end RTL;
```



Define a UNIX Path for the 'VHDL Library'

```
                                  risc_design(CWD)
                   ┌───────────────────────┼───────────┐
                 work/                    rtl/
                   ├ ARCH/                 └ MYREG_rtl.vhd
                   ├ ENTI/
                   ├ MYREG-RTL.syn
                   ├ MYREG.mr
                   └ MYREG.syn
```

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> define_design_lib WORK -path ./work
dc_shell-xg-t> read_vhdl rtl/MYREG_rtl.vhd
```

To keep your CWD structure relatively 'clean' you can redirect the 'VHDL Design Library' to be stored in a separate UNIX directory.

2- 9

Constraining and Compiling Unmapped RTL

```
risc_design(CWD)
├── cons/
│   └── myreg.con
└── rtl/
    └── MYREG_rtl.v
```

TCL: Executes a script

Discussed later

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog rtl/MYREG_rtl.v
dc_shell-xg-t> source cons/myreg.con[1]
dc_shell-xg-t> check_timing
dc_shell-xg-t> compile -boundary -scan -map high
  Error: Could not read the following target libraries:
  your_library.db
```

A 'target' technology library is required to synthesize RTL to gates!

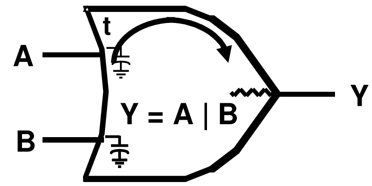
2- 10

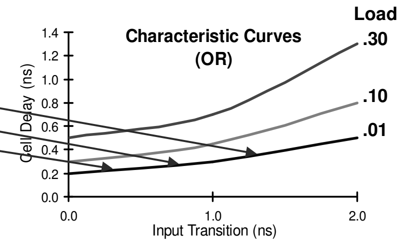
[1] May see Warnings/Errors after executing this step if the constraints refer to library elements.

Example Technology Library

Example of a cell description in *.lib* format

```
cell ( OR2_4x ) {                                       Cell name
    area : 8.000 ;                                      Cell Area
    pin ( Y ) {
        direction : 2;                          2 = Output; 1 = Input
        timing ( ) {
            related_pin : "A" ;
            timing_sense : positive_unate ;
            rise_propagation (drive_3_table_1) {
                values ("0.2616, 0.2711, 0.2831,..)
            }
            rise_transition (drive_3_table_2) {
              values ("0.0223, 0.0254, ...)
                . . . .
        function : "(A | B)";                      Pin Y Functionality
        max_capacitance :  1.14810 ;
        min_capacitance :  0.00220 ;           Design Rules for Pin Y
    }
    pin ( A ) {
        direction : 1;
        capacitance : 0.012000;            Electrical Characteristics
                . . . .                    of Pin A
```





2- 11

The technology library source is an ASCII file (known as ".lib" file), which is compiled by Synopsys **Library Compiler** to create a compiled version (known as ".db" file).

Note: In 'XG Mode' of Design Compiler the DESIGN database format is *ddc*, instead of *db* in 'DB Mode'. The compiled library file format remains *.db*.

For a description of the library format, refer to the *Library Compiler User Guide*.

How is the Target Library Used?

- The *target* library is used <u>during compile</u> to create a technology-specific gate-level netlist
- DC optimization selects the smallest gates that meet the required timing and logic functionality
- Default setting: `(printvar target_library)`

```
target_library = your_library.db
```

Non-existent default library name

- The user must specify the <u>actual</u> synthesis library file provided by the silicon vendor or library group

```
set target_library libs/65nm.db
```

TCL: Variable definition

Reserved DC variable

2- 12

Saving the Gate-level Netlist before Exiting

```
                         risc_design(CWD)
   mapped/                 libs/        cons/           rtl/         ....
   └ MYREG_mapped.v        └ 65nm.db    └ myreg.con     └ MYREG_rtl.v
```

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog rtl/MYREG_rtl.v
dc_shell-xg-t> source cons/myreg.con
dc_shell-xg-t> check_timing
dc_shell-xg-t> set target_library libs/65nm.db
dc_shell-xg-t> compile -boundary -scan -map high
dc_shell-xg-t> write -format verilog -output \
                         mapped/MYREG_mapped.v
dc_shell-xg-t> exit
UNIX%
```

Set *target_library* before *compile*

Save the netlist before exiting DC

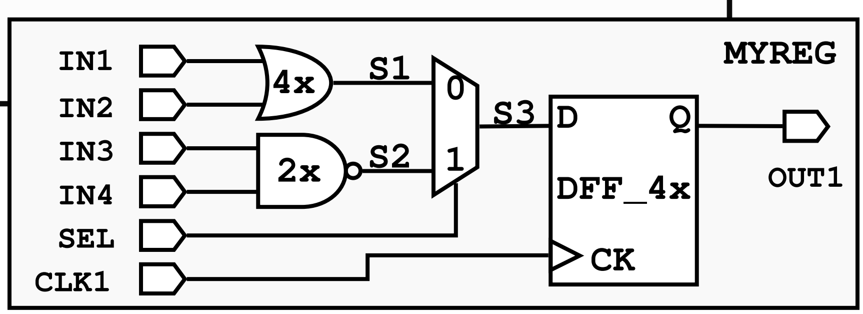
TCL: Command continuation (no blanks!)

2-13

Example resulting netlist after *compile*:

MYREG_mapped.v

```
module MYREG (IN1, IN2, IN3, IN4, SEL, CLK1, OUT1);
  input IN1, IN2, IN3, IN4, SEL, CLK1;
  output OUT1;
  wire S1, S2, S3;

  OR2_4x U1 (.A(IN1), .B(IN2), .Z(S1));
  NAND2_2x U2 (.A(IN3), .B(IN4), .Z(S2));
  MUX21_2x U3 (.A(S1), .B(S2), .S(SEL), .Z(S3));
  DFF_4x OUT1_reg (.D(S3), .CK(CLK1), .Q(OUT1));

endmodule
```



Reading and Analyzing a Netlist

```
                        risc_design(CWD)
                               |
 ---------------------------------------------------------------  ....
     |                  |          |                |
  mapped/             libs/      cons/            rtl/
  └ MYREG_mapped.v    └ 65nm.db  └ myreg.con      └ MYREG_rtl.v
```

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog mapped/MYREG_mapped.v
dc_shell-xg-t> report_constraint -all_violators
 Warning: Can't read link_library file 'your_library.db'.
  ...
 Warning: Unable to resolve reference 'OR2_4x' in `MYREG'.
 Warning: Unable to resolve reference 'NAND2_2x' in `MYREG'.
 Warning: Unable to resolve reference 'MUX21_2x' in `MYREG'.
 Warning: Unable to resolve reference 'DFF_4x' in `MYREG'.
  ...
  ****************************************
  Report : constraint
           -all_violators
  Design:  MYREG
  ...
```

Many commands[1] *auto-link* first!

These warnings do not abort the *report* or *compile* → wasted effort!

2- 14

Many commands[1] will first perform an ‘auto-link’ if the design has not been linked yet. Linking means that DC tries to locate the source of, or ‘resolve’ any instances in the design. Instances can be gates (as in the ‘gate-level netlist example’ above), or sub-blocks (hierarchical modules/entities, hard or soft IP, or DesignWare IP). DC uses the `link_library` variable first to try to resolve the instances, which, just like the `target_library` variable, is set to a non-existent default library *your_library.db*, hence the warnings above. The issue with this auto-linking is that if any problems are encountered, they are considered “warnings” and hence do not abort the command - you end up generating useless reports, or worse, wasting considerable time synthesizing a design with unresolved references. It is therefore highly recommended to explicitly *link* the design immediately after reading it in. This gives you the opportunity to catch and fix any problems before generating reports or compiling the design.

As mentioned earlier, warning and error messages may also appear after sourcing the constraints file, if the constraints refer to cells in technology library. These messages, as well as the warnings after invoking the `report_constraint` command both stem from the same issue: the user must specify a valid `link_library`.

1) The following commands (not all are listed here) automatically link the current design: `check_design/timing`, `compile/_ultra`, `extract`, `group/ungroup`, all `report` commands, constraints and compile directives such as `create_clock`, `set_input/output_delay`, `set_false_path`, etc.

Resolving ‘References’ with `link_library`

- **Default:** `link_library = "* your_library.db"`

"*" represents DC Memory

- **To “resolve” the reference DC:**
  - First looks in DC memory for a matching design name
  - Next looks in the technology library(ies) listed in the *link_library* variable for a matching library cell name
- **The user must replace the default link library with the name of the vendor-provided technology library before *link***

```
set link_library "* $target_library"
```

TCL: “Soft” quotes define a ‘list’ while allowing variable substitution ($var)

2- 15

A ‘reference’ is any gate, block or sub-design that is *instantiated* in your design.
Designs and *Library cells* are key DC database “objects”, to be discussed later in the Unit.

To display the value of the `link_library` variable: `echo $link_library`

TCL: Curly braces, {…}, are treated as “hard quotes”: No variable substitutions or expression calculations are performed. The use of curly braces in the slide example would not work since the value of the `target_library` variable would not be substituted - `link_library` will be literally set to the character string: `* $target_library` instead of `* libs/65nm.db`

TCL: Variable substitution syntax: $*varName*. Variable name can be letters, digits, underscores: a-zA-Z0-9_. Variables do not need to be declared: All of type “*string*” of arbitrary length. Substitution may occur anywhere within a word:

Sample command	Result
`set b 66`	66
`set a b`	b
`set a $b`	66
`set a $b+$b+$b`	66+66+66
`set a $b.3`	66.3 (non-variable character “.” delineates the variable)
`set a $b4`	“no such variable”
`set a ${b}4`	664

Specifying the `link_library` before `link`

```
                    risc_design(CWD)
   |                    |       |           |        ....
mapped/               libs/    cons/       rtl/
 └ MYREG_mapped.v     └ 65nm.db └ myreg.con └ MYREG_rtl.v
```

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog mapped/MYREG_mapped.v
dc_shell-xg-t> set target_library libs/65nm.db
dc_shell-xg-t> set link_library "* $target_library"
dc_shell-xg-t> link
  Loading db file '/../risc_design/libs/65nm.db
  Linking design 'MYREG'
  Using the following designs and libraries:
  ...
  1
dc_shell-xg-t> source cons/myreg.con
dc_shell-xg-t> check_timing
dc_shell-xg-t> report_constraint -all
               ...
```

Explicit `link` - Good practice!

Good News!!

Most `check_`, `report_` and `compile` commands perform an implicit link

2- 16

The purpose of the `link` command is to locate all of the designs and library components referenced in the current design and connect (link) them to the current design.

The `link` command will return a 0 value if it is not able to completely link the design.

Most `check_` , `report_` , `compile` and *optimization* commands will perform an implicit link, so it is, strictly speaking, not necessary to execute an explicit link. It is, however, highly recommended to do so. If a design has a linking problem, you will be able to find the problem explicitly and correct it before executing a report or compile. On the other hand, a linking problem during an implicit link does not abort the command, so you may be compiling, or reporting on, an incomplete design.

There are several additional useful report_ commands which will be discussed in later Units.

Shortening File Name Designations

```
                 risc_design(CWD)
   ----------------------------------------------------- ....
   mapped/                     libs/          cons/
    └ MYREG_mapped.v            └ 65nm.db      └ myreg.con
```

```
read_verilog mapped/MYREG_mapped.v
set target_library libs/65nm.db
set link_library "* $target_library"
link
source cons/myreg.con
check_timing ...
```

Directories can be omitted if they are included in the `search_path` variable

2-17

Using the `search_path` Variable

- **Default search directories:** `(printvar search_path)`

```
search_path =  ". <Install_dir>/libraries/syn
                  <Install_dir>/dw/sim_ver
                  <Install_dir>/dw/syn_ver"
```

"." **represents CWD**

Root DC s/w installation directory

DC library directories, e.g ***gtech.db*** ***standard.sldb***

- **DC looks for specified design and library files in the `search_path` directories**
  - First looks in 'CWD', then the listed directories, in order
- **The user can *add* directories to the default list**

```
set search_path "$search_path mapped libs cons"
```

2- 18

Note: Whenever possible it is recommended that you define the UNIX `search_path` directories as 'relative' paths (e.g. ./mapped or mapped) rather than 'absolute' paths (e.g. /top/my_home/project/mapped). This makes your script files more portable to other UNIX environments.

Modifying the `search_path` Variable

```
                    risc_design (CWD)
   mapped/            libs/      cons/          rtl/
   └ MYREG_mapped.v   └ 65nm.db  └ myreg.con    └ MYREG_rtl.v
```

Search order: left-to-right

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> set search_path "$search_path mapped rtl \
                                   libs cons"
dc_shell-xg-t> read_verilog MYREG_mapped.v
dc_shell-xg-t> set target_library 65nm.db
dc_shell-xg-t> set link_library "* $target_library"
dc_shell-xg-t> link
dc_shell-xg-t> source myreg.con ...
```

These variables need to be specified early on during each DC session (e.g. before *read* or *link*). They can be included in a DC "startup" file instead.

2- 19

Switching Technology Libraries

Old Technology Library	libs/old_1.2um.db
New Technology Library	libs/new_65nm.db
Old Constraints	cons/old.con
New Constraints	cons/new.con

```
set target_library libs/new_65nm.db
set link_library "* $target_library"
read_verilog ORIGINAL_RTL.v
link
source cons/new.con
check_timing
compile –boundary –scan –map high
```

When migrating to a new technology library you should ALWAYS start with the original RTL design.

What if you only have an old *netlist* file?

2- 20

In the example above OLD_NETL.v is assumed to be a flat design with only one top-level module. It is further assumed that the original RTL code is not available, otherwise it would be best to re-compile from RTL, rather than from the gate-level netlist.

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> set search_path "$search_path libs cons"
dc_shell-xg-t> set target_library new_65nm.db
dc_shell-xg-t> set link_library "* old_1.2um.db"
dc_shell-xg-t> read_verilog OLD_NETL.v
dc_shell-xg-t> link
dc_shell-xg-t> source new.con
dc_shell-xg-t> check_timing
dc_shell-xg-t> compile –boundary –scan –map high
dc_shell-xg-t> write –f verilog –out NEW_NTL.v
dc_shell-xg-t> set link_library "* $target_library"   (see note below)
```

1) Note: As long as you do not quit the DC session after *compile* it is not necessary to change the `link_library` variable. The mapped design in DC memory is already linked to the new technology library so any subsequent design analysis will use the new library characteristics. However, once you quit the DC session and subsequently re-invoke DC to read in and analyze the new netlist, you will need to first specify the new technology library as the `link_library`.

Exercise: Migrating a Netlist

Old Technology Library	**libs/old_1.2um.db**
New Technology Library	**libs/new_65nm.db**
Old Constraints	**cons/old.con**
New Constraints	**cons/new.con**

```
set search_path "$search_path ________  ________ "
set target_library ___________________
set link_library   ___________________
read_verilog OLD_NETL.v
link
source ____________________
check_timing
compile -boundary -scan -map high
write -f verilog -out NEW_NTL.v
set link_library ____________________ 1
```

2- 21

In the example above OLD_NETL.v is assumed to be a flat design with only one top-level module. It is further assumed that the original RTL code is not available, otherwise it would be best to re-compile from RTL, rather than from the gate-level netlist.

1) Note: As long as you do not quit the DC session after *compile* it is not necessary to change the `link_library` variable. The mapped design in DC memory is already linked to the new technology library so any subsequent design analysis will use the new library characteristics. However, once you quit the DC session and subsequently re-invoke DC to read in and analyze the new netlist, you will need to first specify the <u>new</u> technology library as the `link_library`.

dc_shell-xg-t> set search_path "$search_path **libs cons**"
dc_shell-xg-t> set target_library **new_65nm.db**
dc_shell-xg-t> set link_library "* **old_1.2um.db**"
dc_shell-xg-t> read_verilog OLD_NETL.v
dc_shell-xg-t> link
dc_shell-xg-t> source **new.con**
dc_shell-xg-t> check_timing
dc_shell-xg-t> compile –boundary –scan –map high
dc_shell-xg-t> write –f verilog –out NEW_NTL.v
dc_shell-xg-t> set link_library "* **$target_library**" (see note below)

Unit Agenda

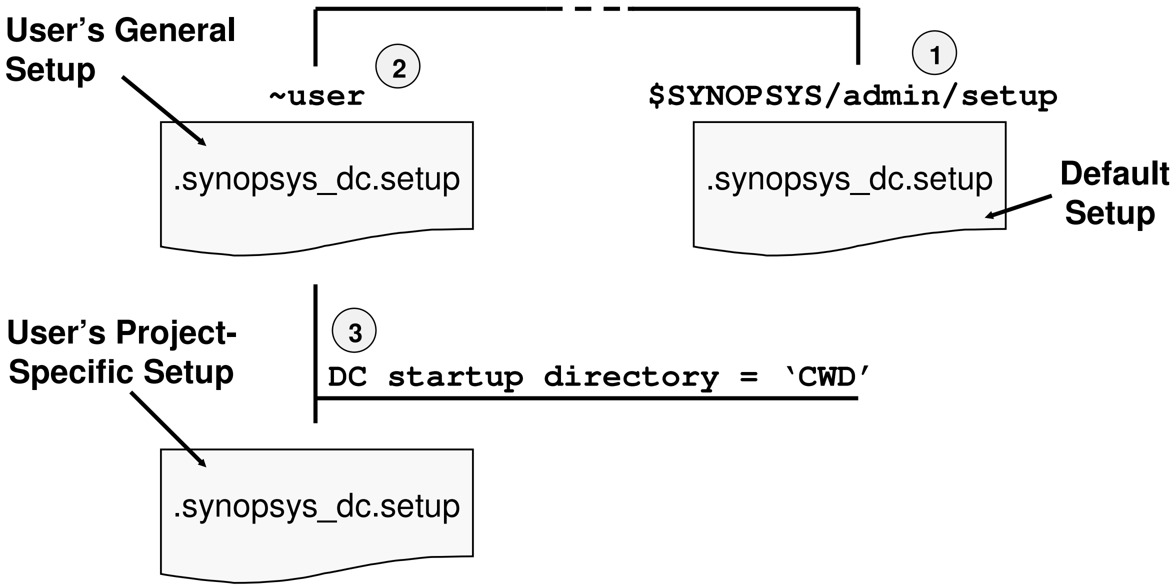
Loading Designs and Libraries

DC Startup File

Loading Hierarchical Designs

2- 22

One Startup File Name – Three File Locations



These files are automatically executed, in the order shown, upon startup of DC.

2- 23

CWD stands for 'current working directory'. We will use this acronym throughout the workshop to represent the UNIX directory in which your DC session was invoked.

Default *…/admin/setup/.synopsys_dc.setup*

```
# .synopsys_dc.setup file in $SYNOPSYS/admin/setup
      .
      .
      .
set target_library your_library.db
set link_library {* your_library.db}
set symbol_library your_library.sdb
set search_path “. <Install_dir>/libraries/syn ...”
      .
      .
```

This file is automatically executed <u>first</u> upon tool startup.

2- 24

Project-specific (CWD) *.synopsys_dc.setup*

TCL: Comment

```
# .synopsys_dc.setup file in project's `CWD'
set search_path "$search_path mapped rtl libs cons"
set target_library 65nm.db
set link_library "* $target_library"
set symbol_library 65nm.sdb; # Contains symbols for
                             # Design Vision GUI

history keep 200
alias h history
alias rc "report_constraint -all_violators"
```

TCL: In-line comment

If present, this file is executed <u>last</u> upon tool startup and overrides previously set variables.

2- 25

Commands/Variables Covered So Far

```
.synopsys_dc.setup
(Executed upon tool start-up)
set target_library 65nm.db
set link_library "* $target_library"
set search_path "$search_path mapped rtl libs cons"
```

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog1 MYREG_rtl.v
dc_shell-xg-t> link
dc_shell-xg-t> source myreg.con
dc_shell-xg-t> check_timing
dc_shell-xg-t> compile -boundary -scan -map high
dc_shell-xg-t> report_constraint -all_violators
dc_shell-xg-t> write -f verilog -out MYREG_mapped.v
```

2- 26

1) For a VHDL file: `read_vhdl MREG_rtl.vhd`

Exercise: Library Setup

```
risc_design(CWD)
├── .synopsys_dc.setup
├── mapped/
│   └── MYREG_mapped.v
├── libs/
│   └── 65nm.db
├── cons/
│   └── myreg.con
└── rtl/
    └── MYREG_rtl.v
```

.synopsys_dc.setup

```
set target_library 65nm.db
set link_library "* $target_library"
set search_path "$search_path mapped libs cons rtl"
```

```
UNIX% cd mapped
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog MYREG_mapped.v
dc_shell-xg-t> link
 Warning: Can't read link_library file 'your_library.db'.
  ...
 Warning: Unable to resolve reference 'OR2_4x' in `MYREG'.
 Warning: Unable to resolve reference 'NAND2_2x' in `MYREG'.
```

What's wrong?

2- 27

Answer: DC was invoked from the *mapped* directory, so the `.synopsys_dc.setup` file was not loaded. To fix the problem the user must exit out of DC, change directories up one level to *risc_design* and then re-invoke DC.

Unit Agenda

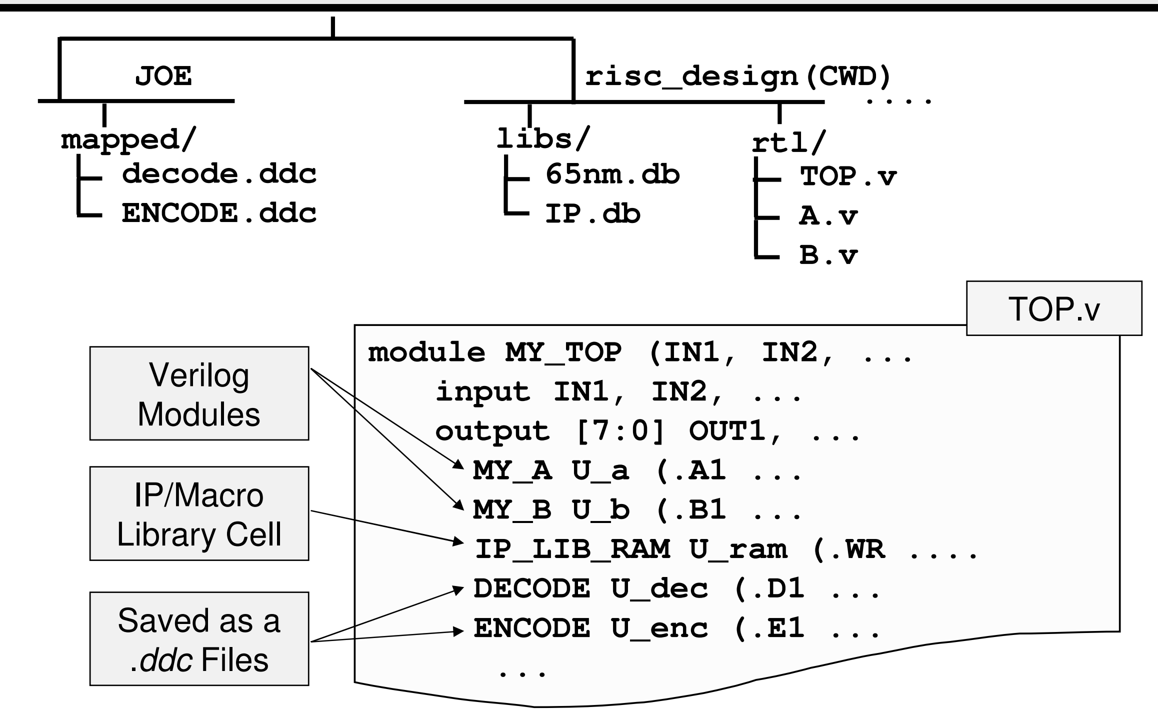
Loading Designs and Libraries

DC Startup File

Loading Hierarchical Designs

2- 28

Example Hierarchical Design



How are different types of instances loaded and resolved?

2- 29

In the example above, the reference:
MY_TOP is a module defined in the RTL file *TOP.v*;
MY_A and *MY_B* are modules defined the the RTL files *A.v* and *B.v*, respectively;
IP_LIB_RAM is an IP or macro cell in the *IP.db* library file;
DECODE and *ENCODE* are compiled sub-designs saved as *ddc* format files *decode.ddc* and *ENCODE.ddc*, respectively.

Reading Hierarchical RTL Designs

```
read_verilog TOP.v
read_verilog A.v
read_verilog B.v
   Current design is 'MY_B'
```

Last file read → current design

```
read_verilog {TOP.v A.v B.v}
   Current design is 'MY_TOP'
```

First file in list → current design

```
read_verilog TOP_hier.v
   Current design is 'MY_A'
```

First module in file → current design

```
risc_design(CWD)
   rtl/
   ├ TOP.v
   ├ A.v
   └ B.v
```

```
risc_design(CWD)
   rtl/
   └ TOP_hier.v
```

2- 30

In the examples above it is assumed that the *search_path* includes the *rtl* directory.

TOP.v

```
module MY_TOP (IN1, IN2, ...
      input IN1, IN2, ...
      output [7:0] OUT1, ...
        MY_A U_a (.A1 ...
        MY_B U_b (.B1 ...
```

A.v

```
module MY_A (A1, A2, OUTA);
 input A1, A2;
      ...
```

B.v

```
module MY_B (B1, B2, OUTB);
       input B1, B2;
            ...
```

TOP_hier.v

```
module MY_A (A1, A2, OUTA);
 input A1, A2;
      ...

module MY_B (B1, B2, OUTB);
 input B1, B2;
      ...

module MY_TOP (IN1, IN2, ...
       input IN1, IN2, ...
       output [7:0] OUT1, ...
        MY_A U_a (.A1 ...
        MY_B U_b (.B1 ...
```

Good Practice: Specify the `current_design`

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
source TOP.con ...
```

Good practice: Specify the `current_design` before `link`

2- 31

Good Practice: `check_design` after `link`

- **`check_design` checks your current design for *connectivity* and *hierarchy* issues, for example:**
  - Missing ports or unconnected input pins
  - Recursive hierarchy or multiple instantiations
- **Issues warnings or errors**
  - Any error returns a 0 value[1]

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
check_design
source TOP.con ...
```

Good practice:

`check_design` after `link`

2- 32

The check_design command, just like the link command, returns a ‘1’ or ‘0’ value to indicate if the action was completed without, or with any serious problems, respectively.

If, instead of executing the above “run” commands interactively, you create a “run script” which is executed in batch mode, be aware that any errors reported by these commands do not abort the run. Your script will continue executing, and may invoke a long compile run on a design with serious problems. You can take advantage of this 1/0 return value to control your command flow, as shown below:

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
if {[link] ==0} {
    echo "Linking Error"
    exit; # Exits DC if a serious linking problem is encountered
}
if {[check_design] ==0} {
    echo "Check Design Error"
    exit; # Exits DC if a check-design error is encountered
}       ; # Script continues to execute if NO problems encountered
source TOP.con
compile -boundary -scan -map high
  ...
```

Resolving IP or Macro Library Cells

TOP.v

```
module MY_TOP (IN1, IN2, ...
   input IN1, IN2, ...
   output [7:0] OUT1, ...
     MY_A U_a (.A1 ...
     MY_B U_b (.B1 ...
     IP_LIB_RAM U_ram (.WR ....
      ...
```

IP Library Cell

```
risc_design(CWD)
├── libs/
│   ├── 65nm.db
│   └── IP.db
├── rtl/
│   ├── TOP.v
│   ├── A.v
│   └── B.v
└── ....
```

```
set link_library "* $target_library IP.db"
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
check_design
source TOP.con ...
```

Include IP or Macro-cell libraries in the `link_library`.
IP cells will be resolved during link.

2- 33

Notice that the *target_library* variable remains unchanged. The *target_library* usually contains only one library file, unless the basic logic cells are split up into separate files (not common).

The most common scenario where the *target_library* contains multiple files is when performing leakage or static power optimization. In this case the *target_library* will contain high- and low-Vth libraries. This is discussed in a later section.

Reading *.ddc* Design Files

TOP.v

```
module MY_TOP (IN1, IN2, ...
   input IN1, IN2, ...
   output [7:0] OUT1, ...
     ....
     DECODE U_dec (.D1 ...
     ENCODE U_enc (.E1 ...
     ...
```

```
JOE                risc_design
 mapped/
   ├ decode.ddc
   └ ENCODE.ddc
```

```
set search_path "$search_path ../JOE/mapped"
set link_library "* $target_library IP.db"
read_verilog {A.v B.v TOP.v}
read_ddc {decode.ddc ENCODE.ddc}[1]
current_design MY_TOP
link
```

The recommended method for loading .ddc files into DC memory is by explicitly reading in the file(s).

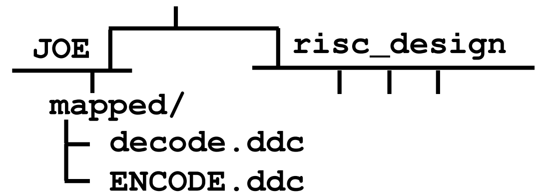
2- 34

[1] .ddc files are created with `write -format ddc -output file_name.ddc`, which will be discussed next.

Auto-loading *.ddc*: Not recommended!

TOP.v

```
module MY_TOP (IN1, IN2, ...
   input IN1, IN2, ...
   output [7:0] OUT1, ...
      ....
     DECODE U_dec (.D1 ...
     ENCODE U_enc (.E1 ...
      ...
```




```
set search_path "$search_path ../JOE/mapped"
set link_library "* $target_library IP.db"
read_verilog {A.v B.v TOP.v}
read_ddc decode.ddc
current_design MY_TOP
link
```

`link` only auto-loads *ddc*, not Verilog or VHDL files.
The file name must be: *design_name*[1].ddc.

2- 35

In general it is better to explicitly read in any design files that are needed. This reduces the risk of DC inadvertently auto-loading an un-intended design, or missing a design because the file name did not exactly match the auto-loading naming convention.

[1] Case-sensitive! If the file was called *DECODE.ddc* instead of *decode.ddc* then the `link` command would resolve this design as well.

Saving the *ddc* Design <u>Before</u> `compile`

```
risc_design(CWD)
├── unmapped/
│   └── MY_TOP.ddc
└── rtl/
    ├── TOP.v
    ├── A.v
    └── B.v
```

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
check_design
write -format ddc -hier -output unmapped/MY_TOP.ddc
source TOP.con
check_timing
compile -boundary -scan -map high
```

The *read* command takes RTL code and builds a 'GTECH' design in DC memory – i.e. translates into <u>unmapped ddc</u> format

- RTL-to-ddc translation of large designs may take some time
- May need to re-read the un-compiled design in the future
- Good practice: Save unmapped *ddc* – *read_ddc* is faster[1]

2- 36

There are two methods for saving hierarchical designs. The first method is the more common approach. In both examples the current design is *MY_TOP*

<u>One</u> file containing the <u>entire</u> design hierarchy:

`write -format ddc` **`-hier`** `-out unmapped/MY_TOP.ddc`

<u>Individual</u> files containing <u>parts</u> of the design hierarchy:

`write -format ddc -out MY_TOP.ddc` (writes the `current_design` by default)
`write -format ddc MY_A -out MY_A.ddc`
`write -format ddc MY_B -out MY_B.ddc`

[1] If the original RTL code is updated you will need to re-read the RTL and re-save the ddc file.

Saving the *ddc* Design <u>After</u> `compile`

```
risc_design(CWD)
├── mapped/
│   ├── MY_TOP.ddc
│   └── MY_TOP_ntl.v
├── unmapped/
│   └── MY_TOP.ddc
└── rtl/
    ├── TOP.v
    ├── A.v
    └── B.v
```

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
check_design
write -format ddc -hier -output unmapped/MY_TOP.ddc
source TOP.con
check_timing
compile -boundary -scan -map high
change_names -rule verilog -hier
write -format verilog -hier -output mapped/MY_TOP_ntl.v
write -format ddc -hier -output mapped/MY_TOP.ddc
```
[1]

Discussed later

The *ddc* format stores the design <u>netlist</u>, <u>constraints</u> and <u>attributes</u>. This is an efficient format to either re-read the design into DC or to read the design into ICC.

2- 37

[1] The Verliog netlist output file is meant for 3rd party downstream tools, e.g Place&Route or simulator. These 3rd party tools may have netlist character restrictions. Because of this it is necessary to invoke the `change_names` command before writing out the netlist (discussed in a later Unit).

ICC = IC Compiler – Performs Power and DFT optimization, placement, clock tree synthesis, routing, DFM, Power using the same timing engine.

Test for Understanding 1/6

1. **Circle the correct statement(s) regarding the `target_` and `link_library` variables:**

   a) Any Macro or IP blocks instantiated in your design should also be included in the `target_library`

   b) During `compile` DC selects the smallest gates that meet timing from the `link_library`

   c) The `link_library` is used to resolve instantiated library cells and the `target_library` is used during `compile`

   d) `link_library` auto-loads .ddc files

2- 38

Answers:

1. C

[To be correct, answer
a) should be `link_library`;
b) should be `target_library`;
d) should be `link`]

Test for Understanding 2/6

2. **Number the items, sequentially starting with "1", to indicate where, and in what order, DC looks to resolve *Z_BOX* during `link`.** (Mark non-applicable items with "N/A")

TOP.v

```
module MY_TOP (A, B,...);
 input A, B, C, ... ;
 output [7:0] OUT1, ... ;
   Z_BOX U3 (.Z1 (Z), ...
   MY_A U1 (.A1 (A), ...
     ...
```

```
set search_path "$search_path rtl ..."
set target_library 65nm.db
set link_library "* $target_library \
                                     IP.db"
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
```

<u>Where DC looks:</u>

Loaded *target library*	____
CWD	____
Files *A.v*, *B.v* and *TOP.v*	____
Loaded *link libraries*	____
Appended `search_path` dir's	____
DC memory	____
Default DC library directories	____

2- 39

Answers:

2.

<u>Where DC looks:</u>

Loaded target library	n/a
CWD	3
Files A.v, B.v and TOP.v	n/a[1]
Loaded link libraries	2
Appended `search_path` dir's	5
DC memory	1
Default DC library directories	4

1) The `link` command does not look in the read-in Verilog files, searching for *module* names that match the reference name. Verilog modules, or VHDL entities, are loaded as 'designs' into DC memory by the `read` command. Auto-load finds these "module" designs by looking in DC memory.

Test for Understanding 3/6

3. **Match each number with a letter from the right column (e.g. 6<u>c</u>) to indicate <u>what</u> DC looks for in each location when resolving *Z_BOX*.** ***"What"*** **items may be used multiple times or not at all.**

<u>Where DC looks:</u>		**<u>What DC looks for:</u>**
Loaded target library	**N/A**	**a. *Z_BOX.ddc***
CWD	**3__**	**b. Library cell *Z_BOX***
Files A.v, B.v and TOP.v	**N/A**	**c. *Z.v***
Loaded link libraries	**2__**	**d. *Z_BOX.v***
Appended `search_path` dir's	**5__**	**e. Design *Z_BOX***
DC memory	**1__**	**f. Library design *Z_BOX***
Default DC library directories	**4__**	**g. Verilog module *Z_BOX***

2- 40

Answers:

3.

<u>Where DC looks:</u>	
Loaded target library	<u>n/a</u>
CWD	<u>3a</u>
Files A.v, B.v and TOP.v	<u>n/a</u>
Loaded link libraries	<u>2b</u>
Appended `search_path` dir's	<u>5a</u>
DC memory	<u>1e</u>
Default DC library directories	<u>4a</u>

Test for Understanding 4/6

TOP.v

```
module MY_TOP (A, B,...);
 input A, B, C, ... ;
 output [7:0] OUT1, ... ;
   Z_BOX U3 (.Z1 (Z), ...
   MY_A U1 (.A1 (A), ...
     ...
```

```
set search_path "$search_path rtl unmppd \
                                 mppd libs"
set target_library 65nm.db
set link_library "* $target_library \
                                 IP.db"
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
```

4. **If *IP.db* contains a cell called *Z_BOX*, and *unmppd* contains a file called *Z_BOX.ddc*, which is auto-loaded?**
5. **If a file called *Zbox.ddc* exists in <u>both</u> the *mppd* and *unmppd* directories, which file is auto-loaded?**

2- 41

Answers:

4. The library cell *Z_BOX*, since it is found first. Once DC resolves a reference it does not search any further.
5. Neither, since DC is looking for a file called *Z_BOX.ddc* ! If both directories contained a file call *Z_BOX.ddc*, then the first file found, in the *unmppd* directory, would be loaded.

Test for Understanding 5/6

6. **To set the current design of a hierarchical design to *TOP*:**
   a) Enter: `current_design TOP`
   b) List file last in: `read_verilog {A.v ... `**`TOP.v`**`}`
   c) Read first: **`read_verilog TOP.v;`** `read_verilog …`
   d) Define *TOP* 'module' last in *TOP_hier.v*
   e) All of the above

7. **What's the advantage of saving <u>unmapped</u> *ddc*?**

2- 42

Answers:

6. A [Answers b, c and d are reversed: whatever is 'last' should be 'first', and vice versa. The point is that, rather than trying to remember these rules, it's much simplere to set the `curent_design` explicitly.]
7. Translating a large Verilog or VHDL RTL design into GTECH, or unmapped ddc, can take some time. If you ever need to re-constrain and/or re-compile the original, un-compiled design, you can save the translation time by reading in the already translated, but as yet unmapped (not compiled) ddc file instead of the original RTL file.

Test for Understanding 6/6

```
read_verilog {A.v B.v T1.v}
current_design TOP
link
write -f ddc -hier -out T2.ddc
source TOP.con
write -f ddc -hier -out T3.ddc
compile -boundary -scan -map high
write -f verilog -hier -out T4.v
write -f ddc -hier -out T5.ddc
write -f verilog -out T6.v
```

8. **What's the difference between:**
   a) T2.ddc and T3.ddc?
   b) T1.v and T4.v?
   c) T3.ddc and T5.ddc?
9. **What module(s) does T6 contain?**
10. **What directory will the T2 – T6 files be written to?**

2- 43

Answers:

8. a) T3.ddc contains the applied constraints and attributes
   T2.ddc is unconstrained
   b) T1.v is the un-synthesized Verilog RTL;
   T4.v is the synthesized Verilog netlist (gate-level)
   c) T5.ddc is mapped - gate-level netlist in ddc format
   T3.ddc is unmapped – GTECH representation of un-synthesized design in ddc format
9. Only the current design is saved as a module called *TOP*.
10. CWD: The first entry in the search_path directory list: ".", the directory where DC was invoked from. To save the files in a different directory, include the full or relative directory path in front of the file name, e.g. `-out` **../Joe/mapped/**T5.ddc .

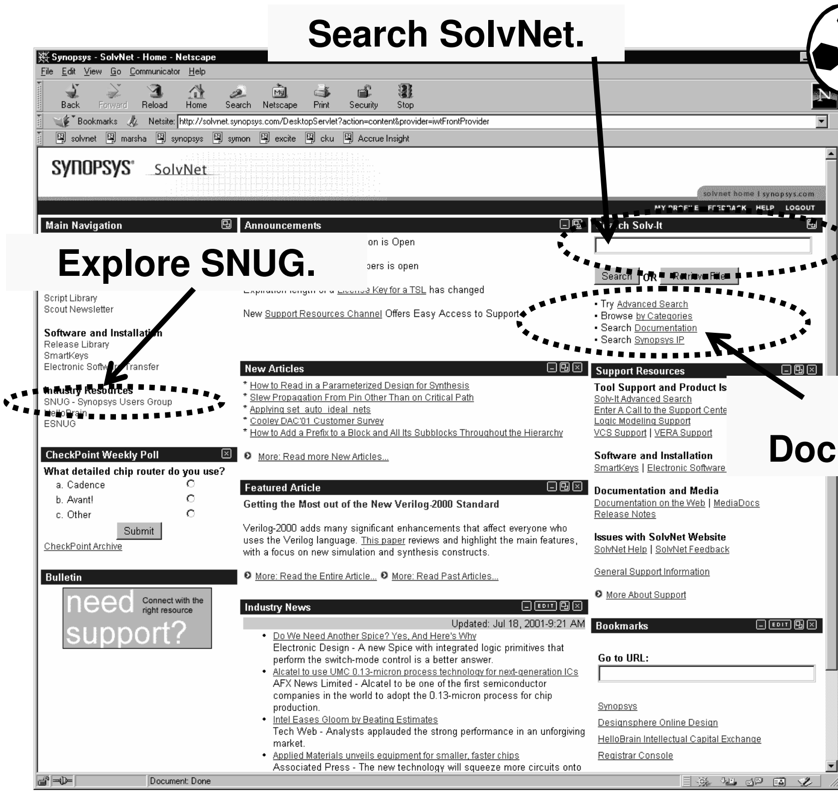
For Further Investigation

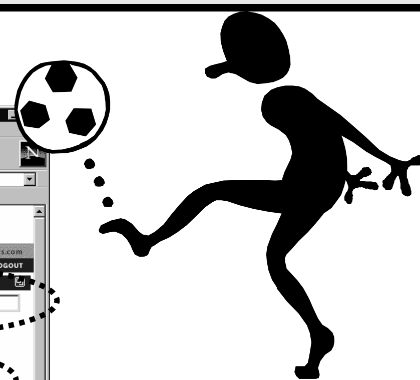
After the workshop, where will you go to find solutions for your Design Compiler problems?

Explore three essential sources of information in lab.

2- 44

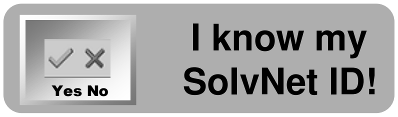
Play During Lab Exercises

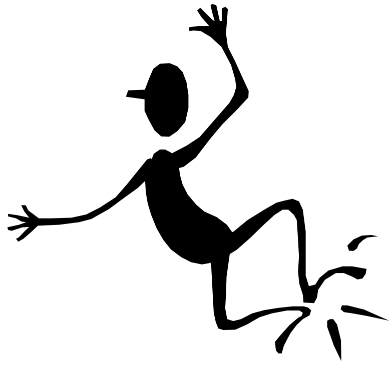




2- 45

What is Your SolvNet ID?





If you do not have a SolvNet ID, please speak with the instructor.

2- 46

Summary: Commands Covered

```
UNIX% cd risc_design
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog1 {A.v B.v TOP.v}
dc_shell-xg-t> current_design TOP
dc_shell-xg-t> link
dc_shell-xg-t> check_design
dc_shell-xg-t> write -f ddc -hier -out unmpd/TOP.ddc
dc_shell-xg-t> source -echo -verbose TOP.con
dc_shell-xg-t> check_timing
dc_shell-xg-t> compile -boundary -scan -map high
dc_shell-xg-t> report_constraint -all_violators
dc_shell-xg-t> change_names -rule verilog -hier
dc_shell-xg-t> write -f verilog -hier -out mpd/TOP.v
dc_shell-xg-t> write -f ddc -hier -out mpd/TOP.ddc
dc_shell-xg-t> exit
```

Invoke DC from the project directory (CWD)

"Good practice" steps

2- 47

[1] Other "read" commands were also be shown – the latter two are shown in the Appendix:

```
read_vhdl
read_ddc
analyze + elaborate
acs_read_hdl
```

Summary: Setup Commands/Variables

.synopsys_dc.setup

```
set search_path “$search_path mapped rtl libs cons”
set target_library 65nm.db
set link_library “* $target_library”
set symbol_library 65nm.sdb

history keep 200
alias h history
alias rc “report_constraint -all_violators”
```

2- 48

Summary: Unit Objectives

You should now be able to prepare a design for compile:

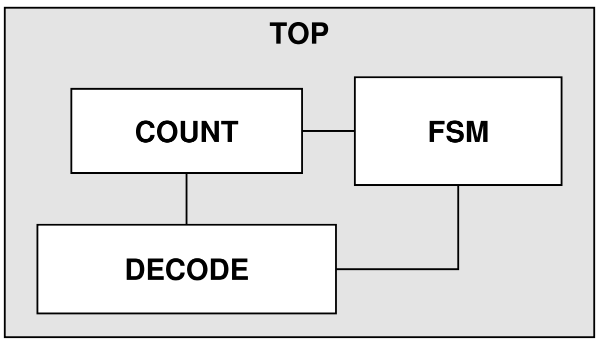
- **Create a DC *setup file* to specify the technology *library file* and *search path* directories**
- **Read in hierarchical designs**
- **Apply a constraints file**
- **Save the design**

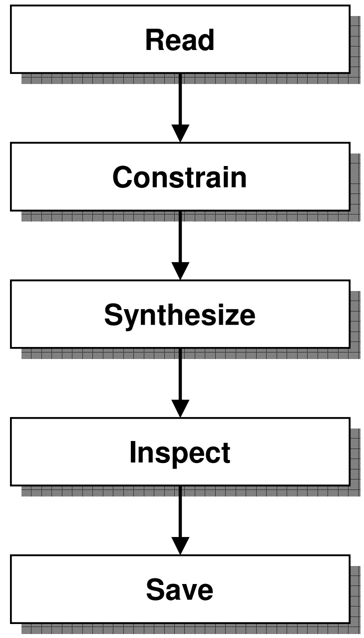
2- 49

Lab 2: Setup and Synthesis Flow

50 minutes

Explore Design Vision and the synthesis flow.





2- 50

Appendix

Alternative Commands for Reading RTL

Reading Designs with `analyze` & `elaborate`

```
risc_design(CWD)
├── rtl/
│   ├── TOP.v
│   └── A.v
├── unmppd/
│   └── MY_B.ddc
└── mppd/
    └── MY_C.ddc
```

TOP.v

```
module MY_TOP (A, B, C, ... );
 input A, B, C, ... ;
 output [7:0] OUT1, ... ;
   MY_A U1 (.A1 (A), ...
   MY_B U2 (.B1 (B), ...
   MY_C U3 (.C1 (C), ...
```

```
set search_path "$search_path rtl unmppd mppd"
analyze -format verilog {A.v TOP.v}
 Compiling source file .../risc_design/rtl/A.v
 Compiling source file .../risc_design/rtl/TOP.v
elaborate MY_TOP
  ...
 Current design is now 'MY_TOP'
 Reading ddc file '.../risc_design/unmppd/MY_B.ddc'.
 Reading ddc file '.../risc_design/mppd/MY_C.ddc'.
```

No need to set the `current_design` or `link`!

2-52

analyze

- Reads source code files (Verilog or VHDL RTL)
- Checks syntax and issues errors/warnings
- Converts both Verilog and VHDL files into intermediate binary format files, placed in CWD
- Can use `define_design_lib` to redirect the files/directories to a sub-directory

elaborate

- Reads the intermediate *.pvl* files and builds the 'GTECH' design in DC memory (unmapped *ddc* format)
- Sets the current design to the specified design
- Links and auto-loads the specified design
- Allows specification of parameter values: *elaborate MY_TOP -parameters "N=8, M=3"*. This is not possible with the *read* command. See example on next page.
- Caution: If available, a .ddc file will be loaded and will over-write an RTL source code design that was explicitly analyzed, if the .ddc file is found in the *search_path* directories.

Note: The `read_vhdl` command ignores *configurations* while `analyze` + `elaborate` does not.

Modifying Parameters with `elaborate`

```
     risc_design(CWD)
  ┌──────────┼──────────┐
rtl/      unmppd/      mppd/
├─ TOP.v  └─ MY_B.ddc  └─ MY_C.ddc
└─ A.v
```

TOP.v

```
module MY_TOP (A, B, C, ... );
 parameter A_WIDTH 2;
 parameter B_WIDTH 4; ...
  input [A_WIDTH-1:0] A;
  input [B_WIDTH-1:0] B; ...
   MY_A U1 (.A1 (A), ...
   MY_B U2 (.B1 (B), ...
```

```
set search_path "$search_path rtl unmppd mppd"
analyze -format verilog {A.v TOP.v}
elaborate MY_TOP -parameters "A_WIDTH=8, B_WIDTH=16"
```

These values override the default values defined in the RTL code

`elaborate` is the only way to change parameter values while reading in a design!

2- 53

Reading Designs with `acs_read_hdl`

```
risc_design(CWD)
├── rtl/
│   ├── TOP.v
│   └── A.v
├── unmppd/
│   └── MY_B.ddc
└── mppd/
    └── MY_C.ddc
```

TOP.v

```
module MY_TOP (A, B, C, ... );
 input A, B, C, ... ;
 output [7:0] OUT1, ... ;
   MY_A U1 (.A1 (A), ...
   MY_B U2 (.B1 (B), ...
   MY_C U3 (.C1 (C), ...
```

```
set search_path "$search_path rtl unmppd mppd"
acs_read_hdl MY_TOP
 Compiling source file .../risc_design/rtl/A.v
 Compiling source file .../risc_design/rtl/TOP.v
 ...
 Current design is now 'MY_TOP'
 Reading ddc file '.../risc_design/unmppd/MY_B.ddc'.
 Reading ddc file '.../risc_design/mppd/MY_C.ddc'.
```

Similar to *analyze* + *elaborate* but no need to list RTL source code files by default

2-54

This command sets the `current_design` to the specified top-level design and links it by reading source code files (Verilog or VHDL RTL) and *ddc* files. Automatically looks for the files containing the top- and sub-level design names (module or entity). It first searches and reads files with *.v* or *.vhd* extensions in the *search_path* directory list (unless the *-hdl_source* command option is specified). It then elaborates the top design, which auto-loads *.ddc* designs found, in left-to-right order, in the *search_path* directory list.

⚠

Because of its "automatic" nature, this command has the potential for unwanted or unexpected results. This can happen for example if you have both Verilog and VHDL files, with similar module/entity names. If a design has been read in from Verilog or VHDL source code, and that design also exists as a *.ddc* file in the *search_path* list, the .**ddc** design replaces the source code design (same behavior as elaborate). Because of this it is highly recommended to direct or focus the command using the appropriate options below, and/or associated variable settings. Please refer to the *man* page for a detailed explanation.

[-hdl_source file_or_dir_list]
[-exclude_list file_or_dir_list]
[-format {verilog | vhdl}]
[-recurse]
[-no_dependency_check]
[-no_elaborate]
[-library design_lib_name]
[-verbose]
[-auto_update | -update file_list]
[-destination destination_dir]

Note: Since there is no direct way to use this command to read in parameterized designs, you can do the following:
`acs_read_hdl -no_elaborate TOP` + `elaborate TOP -param "X=3 Y=5"`

Test for Understanding

11. The commands `analyze + elaborate` can be used:

a) To modify *parameter* values

b) Instead of `read_ddc`

c) Without an explicit `current_design` and `link`

d) Instead of `read_verilog` or `read_vhdl`

e) a, c and d

f) All of the above

12. The nice thing about `acs_read_hdl` is that the only argument you <u>have</u> to specify is the top-level Verilog or VHDL file name – of course you can include more options. True or False?

2- 55

Answers:

11. E. You can not `analyze + elaborate` .ddc files.

12. False. `acs_read_hdl` requires the top-level DESIGN name, not FILE name.

This page was intentionally left blank.

Agenda

DAY 1

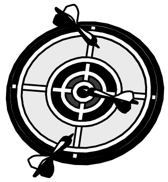
1 Introduction to Synthesis

2 Setting Up and Saving Designs

3 Design and Library Objects

4 Area and Timing Constraints

Unit Objectives



After completing this unit, you should be able to:

- **List the different types of *design* and *library objects***
- **Create a *collection* containing specified object names and type**

3- 2

Commands Covered in this Unit

```
get_ports,  _pins,  _designs,  _cells,  _nets,  _clocks

all_inputs, _outputs, _clocks, _registers

set pci_ports [get_ports "Y??M Z*"]
echo $pci_ports

query_objects $pci_ports
sizeof_collection $pci_ports
set pci_ports [add_to_collection \
                $pci_ports [get_ports CTRL*]]
set all_inputs_except_clk \
           [remove_from_collection [all_inputs] \
             [get_ports CLK]]
filter_collection [get_cells *] "ref_name =~ AN*"
get_cells * -filter "dont_touch == true"

list_attributes -application -class <object_type>
```

3-3

Note: The `get_` and `all_` commands will be covered during the lecture. The remaining commands to access and manipulate *collections* are explained in the Appendix, which the reader is encouraged to study.

Design Objects: Verilog Perspective

Design

Clock[1]

Port

Net

Pin

Cell

```
module TOP (A,B,C,D,CLK,OUT1);
  input A, B, C, D, CLK;
  output [1:0] OUT1;

  wire INV1,INV0,bus1,bus0;

  ENCODER U1 (.AIN (A), . . . .Q1 (bus1));

  INV  U2 (.A (BUS0), .Z( INV0)),
       U3 (.A( BUS1), .Z( INV1));

  REGFILE U4 (.D0 (INV0), .D1 (INV1), .CLK (CLK) );

endmodule
```

3- 4

[1] Clock is a user-defined object within Design Compiler memory.

Design Objects: VHDL Perspective

Design

Clock[1]

Port

Pin

Net

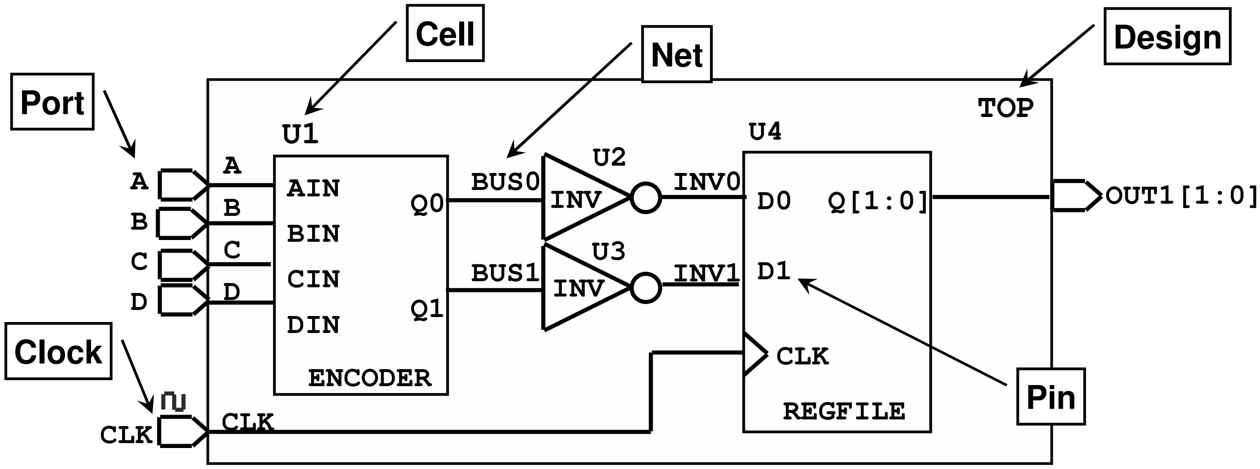
Cell

```
entity TOP is
  port (A, B, C, D, CLK: in STD_LOGIC;
        OUT1: out STD_LOGIC_VECTOR (1 downto 0));
end TOP;
architecture STRUCTURAL of TOP is
 ...
 signal INV1, INV0, BUS1, BUS0: STD_LOGIC;
 begin
  U1: ENCODER port map (AIN => A, Q1 => BUS1);
  U2: INV port map (A => BUS0, Z => INV0);
  U3: INV port map (A => BUS1, Z=> INV1);
  U4: REGFILE port map (D0=>INV0, D1=>INV1,
                        CLK=>CLK);
end STRUCTURAL;
```

3- 5

[1] Clock is a user-defined object within Design Compiler memory.

Design Objects: Schematic Perspective



Pins: {U1/AIN U1/BIN …. U4/Q[0] U4/Q[1]}

Designs: {TOP ENCODER REGFILE}
Cells: {U1 U2 U3 U4}

Why is INV not a design object like ENCODER or REGFILE?

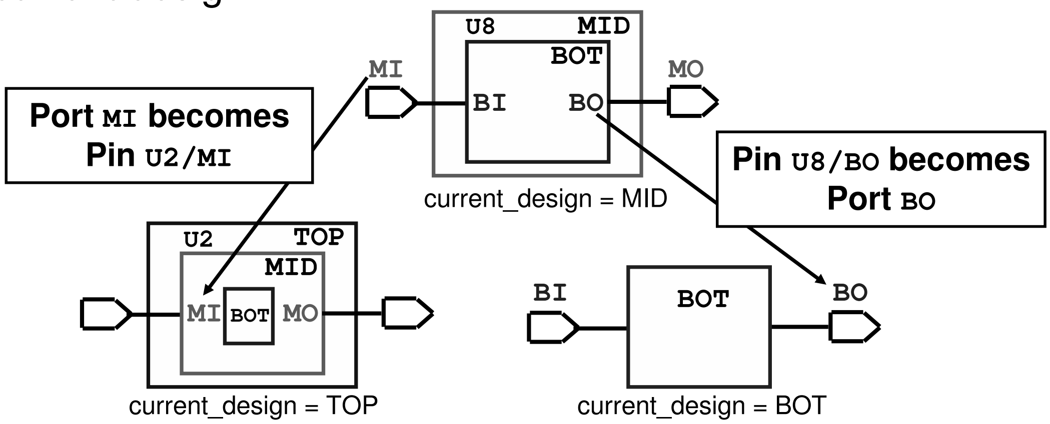
3- 6

Pin are always associated with a specific instance or cell: cell_name/pin_name

ANSWER: INV is a library cell. ENCODER and REGFILE are VHDL entities or Verilog modules.

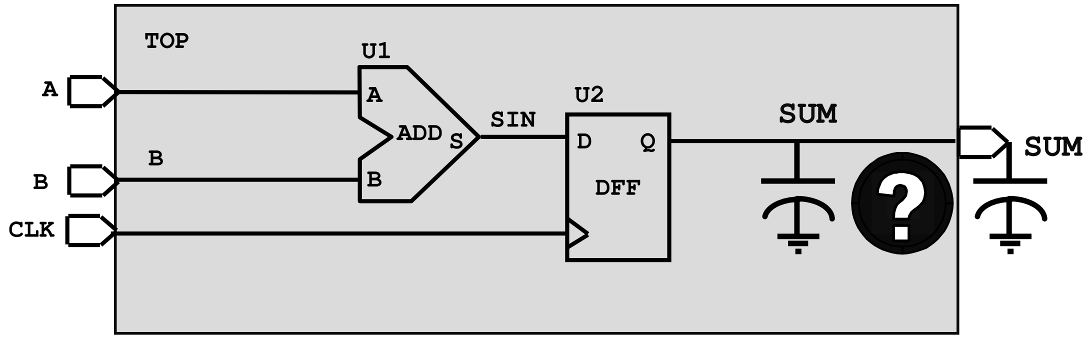
Ports Versus Pins

- **'Ports' are the inputs and outputs of the *current design***
  - They become 'Pins' if the current design moves up to a parent design
- **'Pins' are the inputs and outputs of any cell that is <u>instantiated</u> in the *current design***
  - They become ports if that instantiated design is made the current design

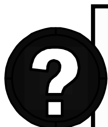


3- 7

Multiple Objects with the Same Name



set_load 5 SUM



Does "SUM" refer to the port or the net object?

Does it matter onto which object DC places the load?

3-8

The SUM argument refers to BOTH the port and the net called SUM – it is ambiguous. If the load is applied to the NET, the user-specified load over-rides any internally-calculated net capacitance (which may be defined by a library wire load model). If the load is applied to the PORT, the load is ADDED to any internally calculated net capacitance, so the total load on the register will be higher, and the delay will be larger.

In the example above DC decides which of the two objects it will apply the load to – it turns out that for this command DC chooses a port object. What if that is NOT what you had intended?

To avoid ambiguity issues it is highly recommended to always specify the object type.

The "get_*" Command

```
dc_shell-xg-t> set_load 5 [get_nets SUM]
```

- **The "`get_*`" commands return objects in the `current_design`, in DC memory, or in libraries:**
  - Can be used stand-alone or embedded in other commands
- **Objects may be used together with ? or * wildcards:**

```
set_load 5 [get_ports addr_bus*]
set_load 6 [get_ports "Y??M Z*"]
```

TCL: Embedded command

- **"`get_*`" commands return a collection of database objects that match the argument(s)**
  - If no matching objects are found, an empty collection is returned

3- 9

Some[1] useful <u>`get` commands:</u>	<u>Returns a collection[2] containing the names of the following design or library objects:</u>
`get_cells`	design *cells* (instances) in the current design, and in sub-blocks with `-hier`
`get_clocks`	*clocks* defined from the current design, or above, on pins/ports at the current design or sub-block level, defined by `create_clock` or `create_generated_clock`. Does not report clocks that were created from within a lower-level design
`get_designs`	*designs* loaded in DC memory (not just in the current design)
`get_libs`	*libraries* loaded in DC memory
`get_lib_cells <libname/cellname>`	*library cells* in the specified library
`get_lib_pins <libname/cellname/pinname>`	*pins* of specified *library cells*
`get_nets`	*nets* at the current design level, and sub-block level with `-hier`
`get_pins <cell/pin>`	input/output *pins* of cells (instances) at the current design level, and sub-block level with `-hier`; pins are always associated with a cell.
`get_ports`	primary input/output/bidir *ports* at the current design level, and sub-block level with `-hier`

1)To see the full list of `get_` commands enter `help get_*`.

2) A collection is a set of objects which is accessible by its collection handle, using special collection commands. More details on collections will be discussed later.

Library Objects

Library Name

File Name

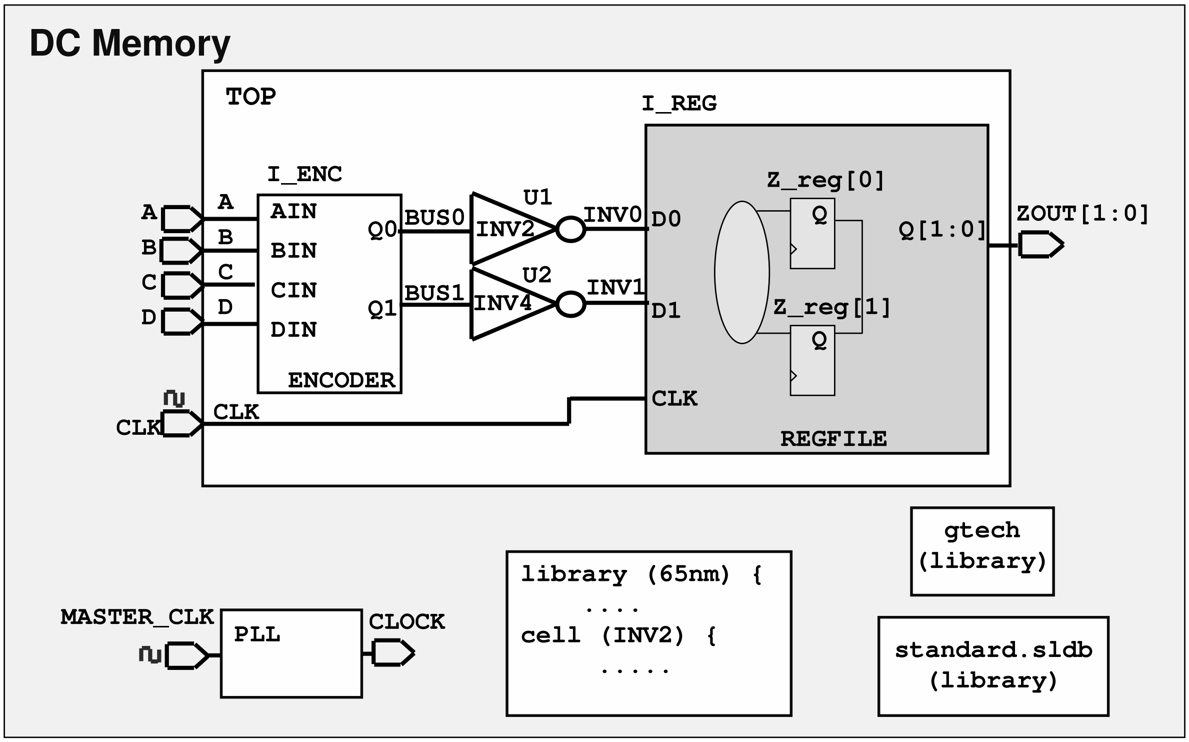
65nm.lib

Library Cell

```
library (vendorX_65nm) {
  .......
cell ( OR2_4x ) {
    area : 8.000 ;
    pin ( Y ) {
        direction : output;
        timing ( ) {
            related_pin : "A" ;
            timing_sense : positive_unate ;
            rise_propagation (drive_3_table_1) {
                 values ("0.2616, 0.2711, 0.2831,..)
            }
            rise_transition (drive_3_table_2) {
              values ("0.0223, 0.0254, ...)
                  . . . .
        function : "(A | B)";
        max_capacitance :  1.14810 ;
        min_capacitance :  0.00220 ;
    }
    pin ( A ) {
        direction : input;
        capacitance : 0.012000;
                  . . . .
```

3- 10

"get_*" Command Exercise 1/2



Use this picture and notes on the previous page to answer the questions on the next page. Assume the *current design* is *TOP*.

3- 11

“get_*” Command Exercise 2/2

1. What does `get_designs *` return? ____________________

2. What does `get_ports {C? Z*}` return? ____________________

3. What does `get_libs` return? ____________________

4. How do you determine all the library cells that begin with INV?

   ____

5. How do you determine all the cells in the entire hierarchy with an underscore “_” in their name? What is returned?

   ____________________ ____________________

6. How do you determine all the Q* pins at the TOP level? What is returned?

   ____________________ ____________________

7. How do you determine all the Q pins inside REGFILE? What is returned?

   ____________________ ____________________

3- 12

ANSWERS

1. {TOP ENCODER REGFILE PLL}
2. {ZOUT[0] ZOUT[1]}
3. {65nm gtech standard.sldb}
4. `get_lib_cells 65nm/INV*` returns {65nm/INV1 65nm/INV2 65nm/INV3 .....}
5. `get_cells *_* -hier` returns {I_ENC I_REG Z_reg[0] Z_reg[1]}
6. `get_pins */Q*` returns {I_ENC/Q0 I_ENC/Q1 I_REG/Q[0] I_REG/Q[1]}
7. `get_pins I_REG/*/Q` returns {I_REG/Z_reg[0]/Q I_REG/Z_reg[1]/Q}

Some Handy `all_*` Commands

- **Gets all input and inout ports of the current design:**

```
dc_shell-xg-t> all_inputs
```

- **Gets all output and inout ports of the current design:**

```
dc_shell-xg-t> all_outputs
```

- **Gets all clocks defined from the current design at the current design level, or below**

```
dc_shell-xg-t> all_clocks
```

- **Gets all register cells in the entire current design's hierarchy:**

```
dc_shell-xg-t> all_registers
```

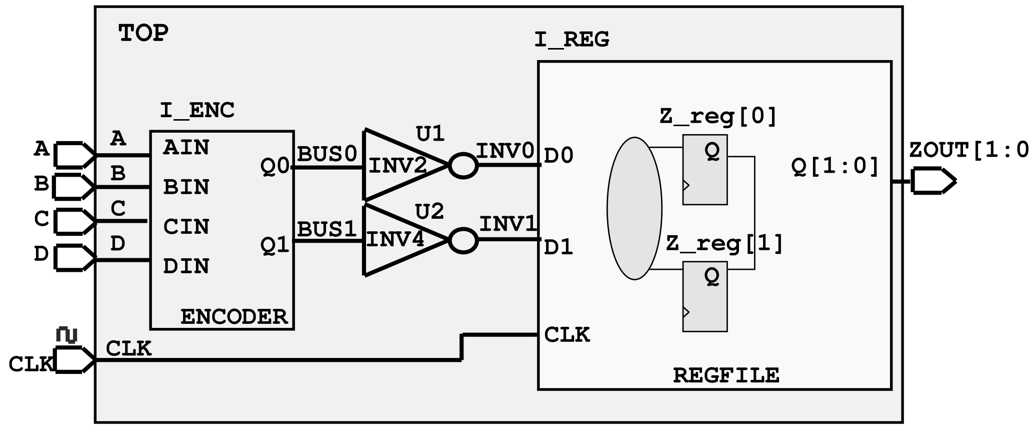
3-13

Some more useful `all_` commands:

```
all_ideal_nets
all_fanin
all_fanout
all_connected
all_dont_touch
all_high_fanout
```

To see the full list of `all_` commands enter `help all_*` .

"all_*" Command Exercise 1/3



Use this picture to answer the questions on the next page.
Assume that the current design is TOP.

3- 14

“all_*” Command Exercise 2/3

1. What does `all_inputs` return?

   ____

2. What does `all_outputs` return?

   ____

3. What does `all_registers` return?

   ____

4. What does `all_inputs C*` return?

   ____

3- 15

ANSWERS

1. {A B C D CLK}
2. {ZOUT[0] ZOUT[1]}
3. {I_REG/Z_reg[0] I_REG/Z-reg[1]}
4. "Error: extra positional option 'C*' (CMD-012)". (`all_` commands do not accept an object name as an argument)

“all_*” Command Exercise 3/3

5. Can you guess what the following returns?

   `remove_from_collection [all_inputs] [get_ports CLK]`

   ____

<u>Homework</u>:

Study the Appendix to learn more about:

- **Accessing and manipulating *collections***
- **TCL syntax**

3- 16

ANSWERS

5. {A B C D}

Summary: Commands Covered

```
get_ports,  _pins,  _designs,  _cells,  _nets,  _clocks

all_inputs, _outputs, _clocks, _registers

set pci_ports [get_ports "Y??M* Z*"]
echo $pci_ports

query_objects $pci_ports
sizeof_collection $pci_ports
set pci_ports [add_to_collection \
                $pci_ports [get_ports CTRL*]]
set all_inputs_except_clk \
           [remove_from_collection [all_inputs] \
             [get_ports CLK]]
filter_collection [get_cells *] "ref_name =~ AN*"
get_cells * -filter "dont_touch == true"

list_attributes -application -class <object_type>
```

3- 17

Note: The `get_` and `all_` commands were covered during the lecture. The remaining commands to access and manipulate *collections* are explained in the Appendix, which the reader is encouraged to study.

Summary: Unit Objectives

You should now be able to:

- **List the different types of *design* and *library objects***
- **Create a *collection* containing specified object names and type**

3- 18

Appendix

Accessing and Manipulating Collections

Objects and Attributes

- **Recap: Designs can contain six different objects**
  - Designs, cells, ports, pins, nets and clocks
- **In order to keep track of circuit functionality and timing, DC attaches many attributes to each of these objects:**
  - Ports can have the following attributes

direction	**driving_cell_rise**	
load	**max_capacitance**	many others …

  - Cells can have the following attributes

dont_touch	**is_hierarchical**	
is_mapped	**is_sequential**	many others ...

3- 20

To see all DC-defined attributes for a particular object type:

```
dc_shell-xg-t> list_attributes -application -class <object_type>
```

where `<object_type>` = design, port, pin, cell, net, clock, lib, reference, cluster or bag.

Since the list of attributes can be quite long it may be easier to re-direct the output to a file, which can then be grepped for a specific string, for example:

```
dc_shell-xg-t> redirect -file cell_attr {list_attributes -application \
                                         -class cell
UNIX% grep dont_ cell_attr | more

....
dont_duplicate_csa                  cell       boolean    a
dont_split_arithmetic               cell       boolean    a
dont_touch                          cell       boolean    a
dont_touch_ref_design               cell       boolean    u
dont_transform_csa                  cell       boolean    a
dont_uniquify                       cell       boolean    a
dont_use                            cell       boolean    a
....
```

Accessing the Synopsys Database

- **Access to DC *objects* in DC-Tcl is achieved through collections - a DC extension to standard Tcl**
- **When executed the `get_` or `all_` command:**
  - Creates a collection of objects, with their attributes
  - Echoes the contents of the collection to the screen
  - Returns a collection ‘handle’ - <u>not</u> a list of objects

TCL: Define a variable

The variable is a ‘collection’

Display only

Collection handle

```
dc_shell-xg-t> set foo [get_ports p*]
{pclk pframe_n pidsel pad[31]...}
dc_shell-xg-t> echo $foo
_sel184
```

TCL: Echoes the argument to the screen

3- 21

Accessing and Manipulating Collections

```
dc_shell-xg-t> set foo [get_ports p*]
{pclk pframe_n pidsel pad[31]...}
dc_shell-xg-t> sizeof_collection $foo
50
dc_shell-xg-t> query_objects $foo
{pclk pframe_n pidsel pad[31]...}
```

```
dc_shell-xg-t> set pci_ports [get_ports DATA*]
{DATA[0] DATA[1] DATA[2] ...}
dc_shell-xg-t> set pci_ports [add_to_collection \
                     $pci_ports [get_ports CTRL*]]
{DATA[0] DATA[1] DATA[2] ... CTRL_A CTRL_B}
```

```
dc_shell-xg-t> set all_inputs_except_clk \
              [remove_from_collection [all_inputs] \
               [get_ports CLK]]
```

3- 22

Collection commands return a <u>collection handle</u>, NOT a list! Standard Tcl list commands (`concat`, `llength`, `lappend`, etc) <u>will not work</u> with the output of a collection command!

```
dc_shell-xg-t> help *collection*
add_to_collection       # Add object(s)
compare_collections     # compares two collections
copy_collection         # Make a copy of a collection
filter_collection       # Filter a collection, resulting
                           in a new collection
foreach_in_collection   # Iterate over a collection
index_collection        # Extract object from collection
remove_from_collection  # Remove object(s) from a collection
sizeof_collection       # Number of objects in a collection
sort_collection         # Create a sorted copy of a collection

dc_shell-xg-t> help *object*
more collection related commands ...
```

Filtering Collections

- **Use the `filter_collection` command to get only objects you are interested in:**

```
filter_collection [get_cells *] "ref_name =~ AN*"
filter_collection [get_cells *] "is_mapped != true"
```

- **The `–filter` option is a nice short-cut:**

```
get_cells * -filter "dont_touch == true"
set fastclks [get_clocks * -filter "period < 10"]
```

- **Relational operators are:**

```
==, !=, >, <, >=, <=, =~, !~
```

3- 23

Description of the examples above:
1. Returns all cells starting with the name "AN"
2. Returns all unmapped cells
3. Returns all cells with the "dont_touch" attribute
4. Returns all clocks with a period smaller than 10

`filter_collection` creates a new collection, or an empty string if no objects match the expression.
The `-filter` option is more efficient, because the collection does not have to be read twice.
Other examples:

```
get_cells -hier -filter "is_unmapped != true"
get_cells -hier -filter "is_hierarchical == true"
```

Iterating Over a Collection

- **Use the `foreach_in_collection` command to iterate over a collection:**

```
foreach_in_collection cell [get_cells -hier * -filter \
                            "is_hierarchical == true"] {
echo "Instance [get_object_name $cell] is hierarchical"
}
```

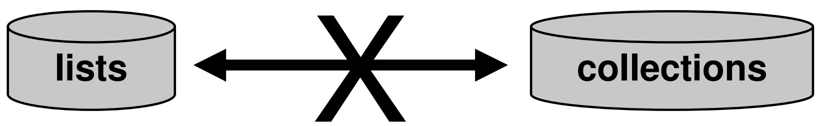
```
Instance I_Ablock is hierarchical
Instance I_CONTROL is hierarchical
Instance I_Bblock is hierarchical
...
```

3- 24

`get_object_name` is more useful in this case than `query_objects`: the latter returns {I_Ablock} while the former returns I_Ablock.

Collection Versus Tcl List Commands

- ***Tcl lists*** **are structures to store** ***user-defined*** **data**
- ***Collections*** **are used to access** ***database*** **data**
  - Collections are more memory efficient for this purpose
- **List commands should not be used on collections and vise versa, for example:**
  - Use `foreach` to iterate through a *list*
  - Use `foreach_in_collection` to iterate through a *collection*



3- 25

The above is a strong recommendation. DC does allow some mixing of lists and collections, but it is not recommended.

For example, the following is allowed:

```
set port_col [list [get_ports a*] [get_ports b*]]
```

`port_col`: is a list with two collections. This list may be passed to other collection manipulation commands.

It is better to convert the command to this:

```
set port_col [get_ports "a* b*"]
```

This page was intentionally left blank.

Agenda

DAY 1

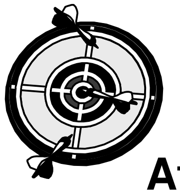
1 Introduction to Synthesis

2 Setting Up and Saving Designs

3 Design and Library Objects

4 Area and Timing Constraints

Unit Objectives

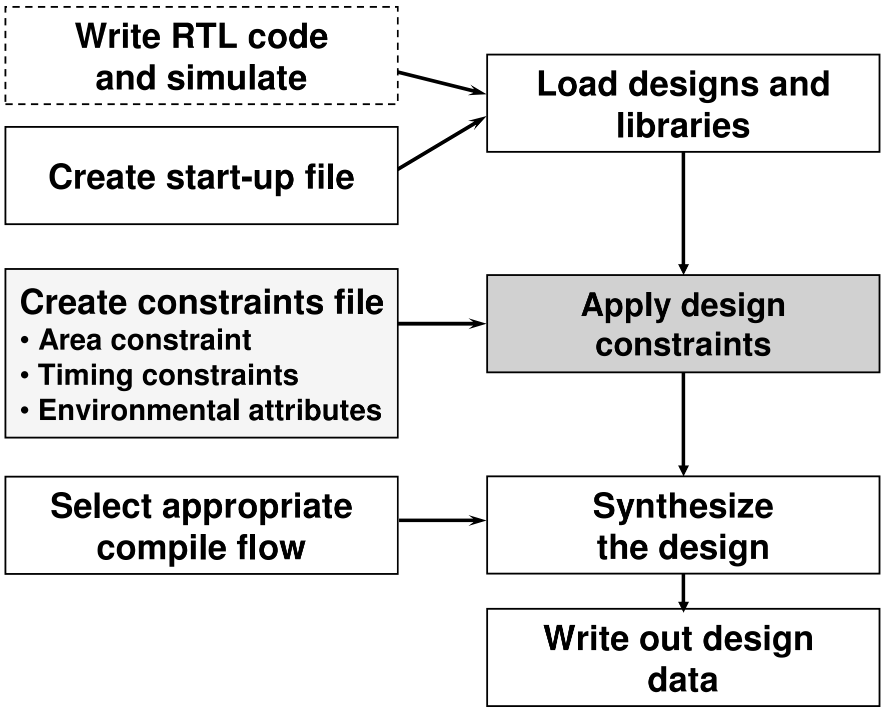


After completing this unit, you should be able to:

- **Constrain a design for area**
- **Constrain a design for <u>setup</u> timing, under the following conditions:**
  - The design’s specs are given or known
  - Can be block- or chip-level design
  - Single clock, single cycle environment
  - Default design scenario – minimal command options
- **Create and execute a constraints file**

4- 2

RTL Synthesis Flow



4-3

Commands To Be Covered (1 of 2)

TOP.con

```
# Area and Timing Constraints
#

reset_design;  # Good practice step

set_max_area 245000
create_clock -period 2 [get_ports Clk]
create_clock -period 3.5 -name V_Clk; # VIRTUAL clock
set_clock_uncertainty -setup 0.14 [get_clocks Clk]
set_clock_latency -source -max 0.3 [get_clocks Clk]
set_clock_latency -max 0.12 [get_clocks Clk]
set_clock_transition 0.08 [get_clocks Clk]
set_input_delay -max 0.6 -clock Clk [all_inputs]
set_input_delay -max 0.5 -clock V_Clk [get_ports "A C F"]
set_output_delay -max 0.8 -clock Clk [all_outputs]
set_output_delay -max 1.1 -clock V_Clk [get_ports "OUT2 OUT7]
```

4- 4

Commands To Be Covered (2 of 2)

RUN.tcl

```
# Run script
#
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
redirect -tee -file precompile.rpt {link}
redirect -append -tee -file precompile.rpt {check_design}
redirect -append -tee -file precompile.rpt {source -echo -ver TOP.con}
redirect -append -tee -file precompile.rpt {report_port -verbose}
redirect -append -tee -file precompile.rpt {report_clock}
redirect -append -tee -file precompile.rpt {report_clock -skew}
redirect -append -tee -file precompile.rpt {check_timing}
 ...
```

```
UNIX% dc_shell-t -f RUN.tcl | tee -i my.log
```

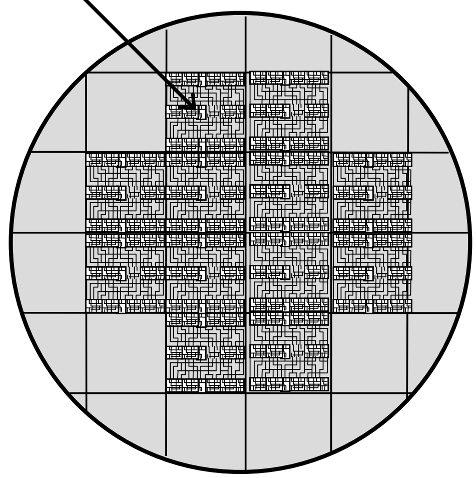
```
help *clock
help -verbose create_clock
create_clock -help
printvar *_library
man target_library
```

4- 5

Specifying an Area Constraint

```
dc_shell-xg-t> read_verilog MY_DESIGN.v
dc_shell-xg-t> current_design TOP_CHIP_or_BLOCK
dc_shell-xg-t> link
dc_shell-xg-t> set_max_area 245000
```

- **Area unit is defined by the library supplier – it's not in the library so ask!**
  - 2-input NAND gates
  - Transistors
  - mil^2, mm^2 or μm^2
- **How do you determine what value to use?**
  - From the spec or project lead
  - If migrating to a newer technology use a smaller % of the old design size
  - Estimate based on experience



? Is `set_max_area 0` acceptable ?

4-6

The set_max_area and the up-coming constraints are applied to the "current design". The current design is implicitly set by Design Compiler (DC) from the last module or entity name in your RTL code. This "rule" was not always this way. The current design used to be determined by the first module/entity in earlier versions of DC. Because this "rule" can change from one version to another, it is recommended to <u>explicitly</u> set the current design rather than relying on The area units are not shown by `report_lib`, you have to ask the ASIC vendor.

Design Compiler will perform minimal area optimization unless an area constraint is set.

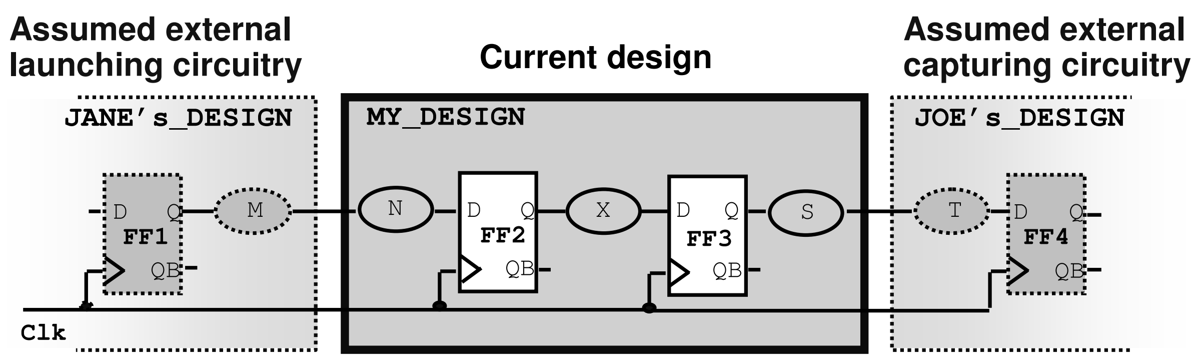
Design Compiler will quit area optimization when the area constraint is reached. If a constraint is too aggressive, or if 0 is used, Design Compiler will perform area optimizations until it reaches a point of diminishing returns, at which time optimization will halt. This may increase run time but will not impact DC's ability to meet timing constraints. Timing constraints always have higher priority over area constraints. If run-time is a concern, try to apply as realistic an area goal as possible. If run-time is not a concern, it's acceptable to set it to 0.

Specifying Setup-Timing Constraints

- **Objective: Define setup timing constraints for all paths within a sequential design**
  - All input logic paths (starting at input ports)
  - The internal (register to register) paths
  - All output paths (ending at output ports)
- **Under the following conditions:**
  - You are given the design's specs
  - Block- or chip-level design
  - Single clock, single cycle or environment

4- 7

Default Design Scenario

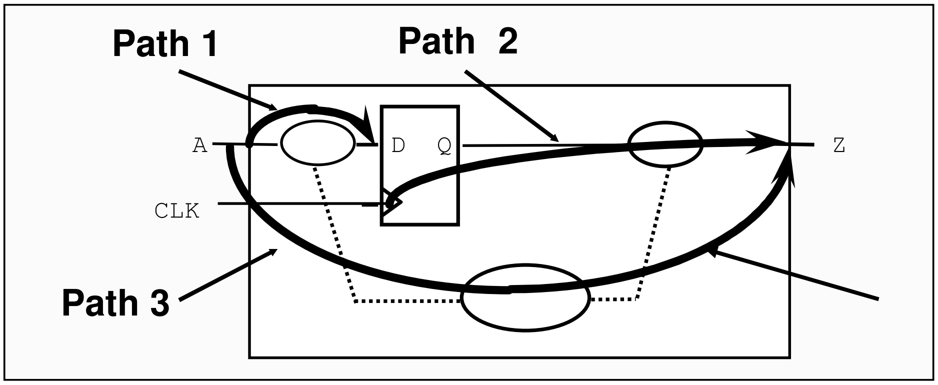


Design Compiler assumes a "synchronously-clocked" environment. By default:

- Input data arrives from a pos-edge clocked device
- Output data goes to a pos-edge clocked device

4- 8

Timing Analysis During/After Synthesis



DC breaks designs into timing paths, each with a:

- **Startpoint**
  - Input port
  - Clock pin of Flip-Flop or register
- **Endpoint**
  - Output port
  - Any input pin of a sequential device, except clock pin[1]

4- 9

[1]Examples of non-clock input pins of a sequential device: D (Data), S (Set) or R (Reset), E (Enable).

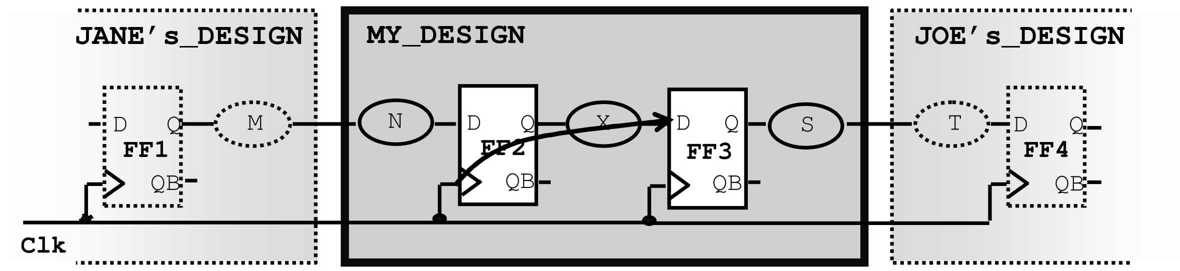
Static Timing Analysis determines if a circuit meets timing constraints during and after synthesis This involves three main steps:

1) The design is broken down into sets of timing paths

2) The delay of each path is calculated

3) Path delays are compared against expected arrival times to determine if constraints have been met or not

Constraining Register-to-Register Paths



What information must <u>you</u> provide to constrain all the register-to-register paths in MY_DESIGN for setup time?

4-10

The duty cycle matters only if your registers are clocked by both positive and negative edges of the clock. In the example above both registers are positive-edge-triggered so the duty cycle does not matter. The only information the user must supply is the clock period. DC retrieves the setup time of FF3 from the technology library.

DC automatically calculates the maximum delay for the path, starting at the clock pin of FF2 and ending at the D-pin of FF3, as 10 – 1 = 9ns.

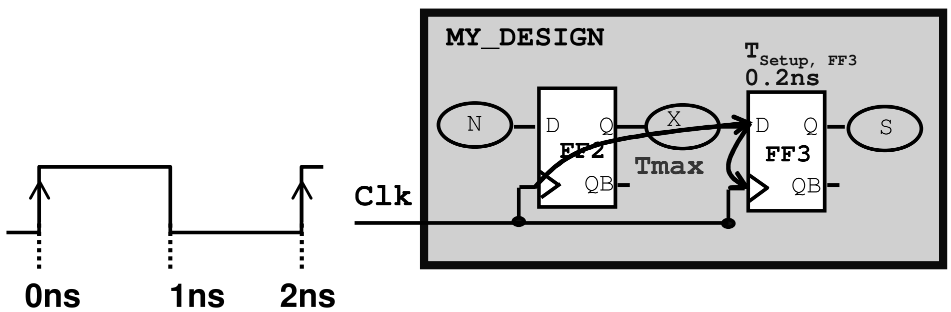
Constraining Reg-to-Reg Paths: Example

Spec:

Clock Period = 2ns

Unit of time is 1ns in this example. Defined in the technology library.

```
create_clock -period 2 [get_ports Clk]
```



What is the maximum delay requirement Tmax for the register-to-register path through X in the MY_DESIGN? ____________________

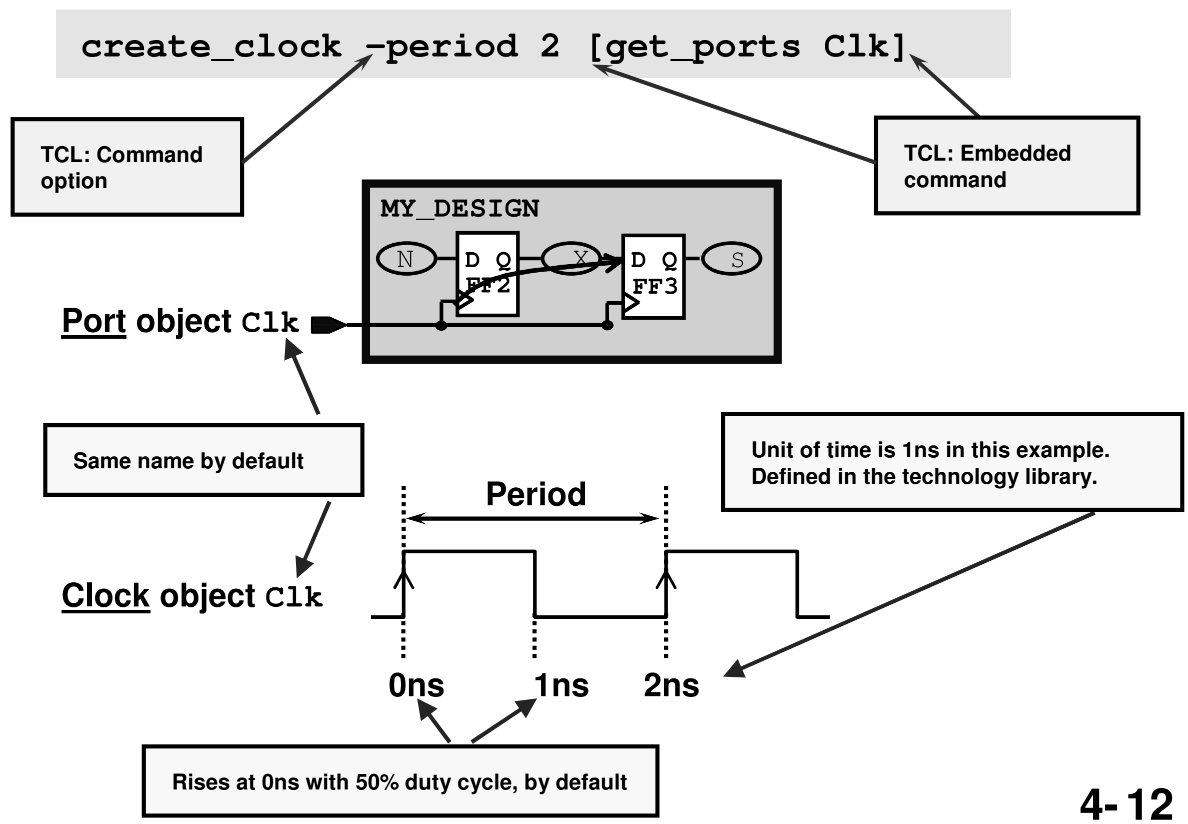
4- 11

Treg-to-reg, max = P – Setup_Time (assuming 0 clock skew) = 2 – 0.2 = 1.8ns

Unless explicitly defined with the `-waveform` option, the clock object that is created starts off going high at 0ns, comes down at 50% of the clock period, and repeats every clock period. The clock object's name is the same as the port/pin object to which it is assigned, unless explicitly defined by the `-name` option.

The unit of time is defined in the technology library. It is commonly 1 nano-second, as in the example above, but it may also be defined differently, for example: 0.1 nano-second, 10 pico-seconds, or 1 pico-second. You should verify the unit of time by looking at the top of the report generated by `report_lib [library_name]` , or by get_attribute lib_name time_unit_name.

`create_clock` Required Arguments



Unless explicitly defined with the `-waveform` option, the clock object that is created starts off going high at 0ns, comes down at 50% of the clock period, and repeats every clock period. The clock object's name is the same as the port/pin object to which it is assigned, unless explicitly defined by the `-name` option.

The unit of time is defined in the technology library. It is commonly 1 nano-second, as in the example above, but it may also be defined differently, for example: 0.1 nano-second, 10 pico-seconds, or 1 pico-second. You should verify the unit of time by looking at the top of the report generated by `report_lib [library_name]` .

Default Clock Behavior

- **Defining the clock in a single-clock design constrains all timing paths <u>between</u> registers for <u>single-cycle</u>, <u>setup</u> time**
- **By default the clock rises at 0ns and has a 50% duty cycle**
- **By default DC will not "buffer up" the clock network, even when connected to many clock/enable pins of flip-flops/latches**
  - The clock network is treated as "ideal" - infinite drive capability
    - Zero rise/fall transition times
    - Zero skew
    - Zero insertion delay or latency
  - Estimated skew, latency and transition times can, and should be modeled for a more accurate representation of clock behavior

4- 13

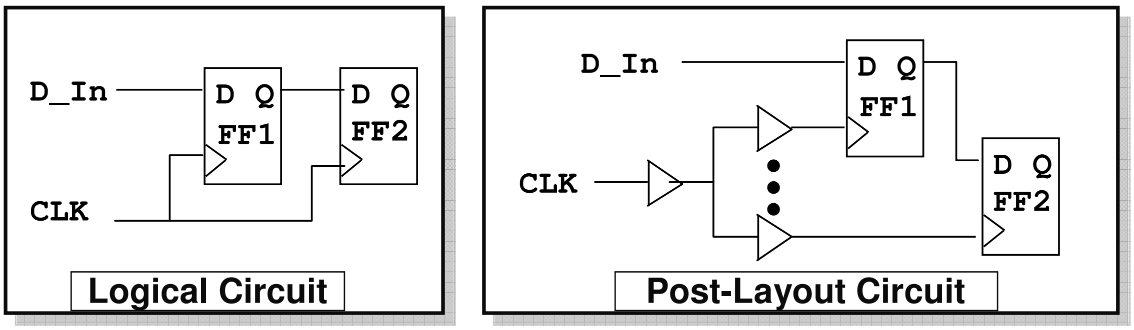
Load balancing is also known as "design rule optimization", which entails meeting maximum/minimum transition, capacitance and/or fanout "design rules". These design rules are defined in the technology library and apply to non-clock nets. Clock nets are exempt from meeting these design rules.

Skew balancing is also known as "clock tree synthesis" (CTS). Design Compiler does not perform skew balancing or CTS.

The recommended approach to dealing with clocks is to model estimated skew, latency and transition times during synthesis (using commands that will be covered in a later Unit). CTS is then performed after synthesis, with a "back-end" or "physical" tool like Physical Compiler, IC Compiler, Astro, or 3rd party tool.

Modeling Clock Trees

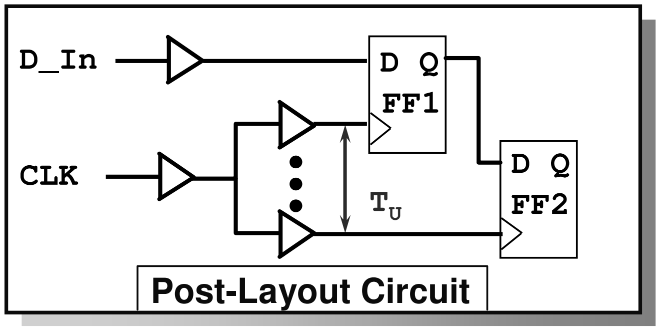
- **Design Compiler is NOT used to synthesized clock buffer trees**
- **Clock tree synthesis is usually done by a physical or layout tool, based on actual cell placement**



? **What clock tree effects need to be taken into account by the synthesis tool, prior to layout?**

4- 14

Modeling Clock Skew



Uncertainty models the maximum delay difference between the clock network branches, known as clock skew, but can also include clock jitter and margin effects:

```
set_clock_uncertainty -setup TU [get_clocks CLK]
```

Pre-Layout: clock skew + jitter + margin

placed on clock objects

4- 15

The default clock uncertainty is zero.

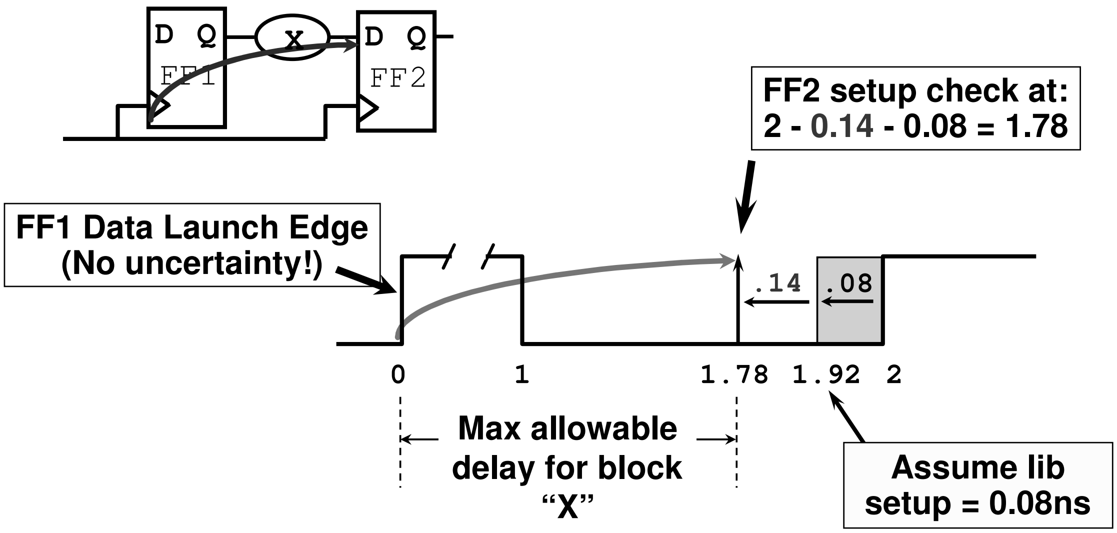
Uncertainty can also be defined between clock domains with as follows:

```
set_clock_uncertainty -setup TU -from <Clk1> -to <Clk2>
```

`set_clock_uncertainty` and Setup Timing

Example:

```
create_clock -period 2 [get_ports CLK]
set_clock_uncertainty -setup 0.14 [get_clocks CLK]
```



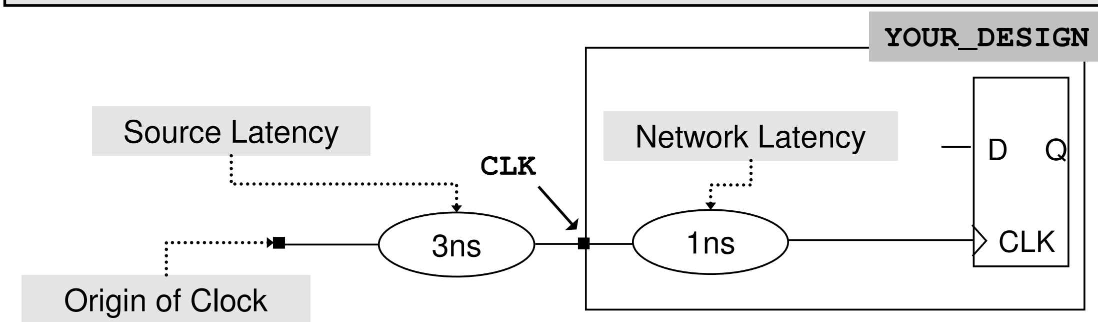
4- 16

By default, the command "`set_clock_uncertainty`" assigns the same value to the setup and the hold uncertainty, unless either the –setup or the –hold switch are used. This allows the design to be synthesized with extra "clock period margin" for setup timing, which does not apply to hold timing.

Modeling Latency or Insertion Delay

- **Network latency models the average 'internal' delay from the `create_clock` port or pin to the register clock pins**
- **Source latency models the delay from the actual clock origin to the `create_clock` port or pin:**
  - Used for either ideal or propagated clocks (post layout)

```
create_clock -period 10 [get_ports CLK]
set_clock_latency -source -max 3 [get_clocks CLK]
set_clock_latency -max 1 [get_clocks CLK] ;# pre layout
#set_propagated_clock [get_clocks CLK] ;# post layout
```



4- 17

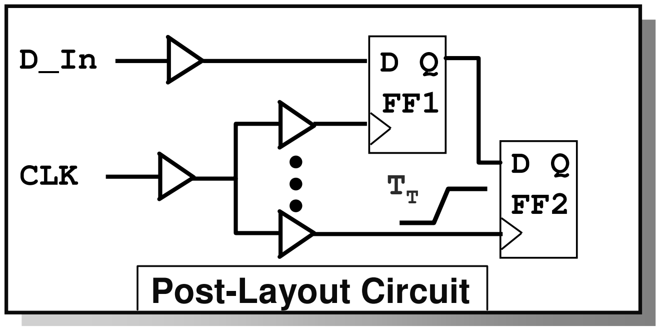
One of the benefits of this is for propagated clocks. Propagated clocks will calculate network latency. In addition you can now model source latency. The total latency to the register will be the source + network latency.

If you do not use the -source option with `set_clock_latency`, then it represents network latency.

You can model rise, fall, min and max numbers for both network and source latency.

Useful when clock generation circuitry is not part of your design, or for derived clocks.

Modeling Transition Time

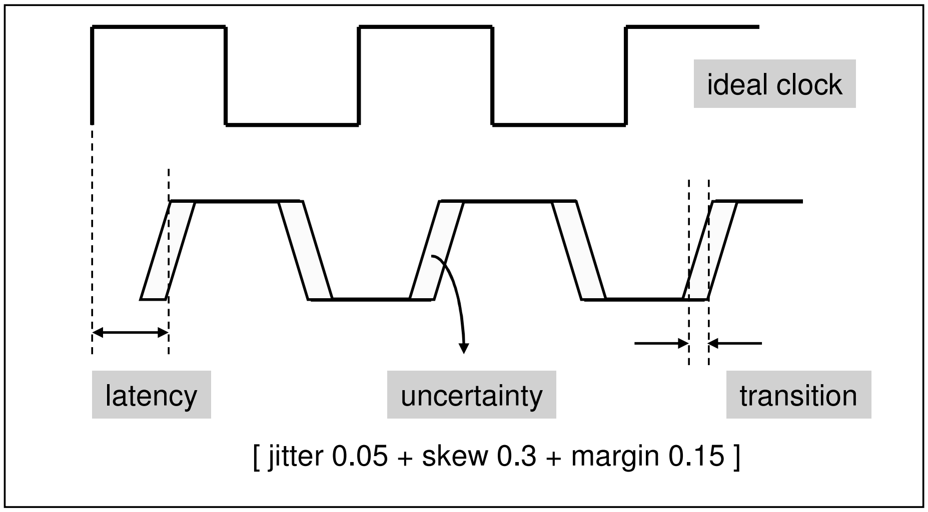


Transition models the rise and fall times of the clock waveform at the register clock pins:

```
set_clock_transition TT [get_clocks CLK]
```

4- 18

Pre/Post Layout Clock



Synthesis Constraints

```
reset_design
create_clock -p 5 -n MCLK Clk
set_clock_uncertainty 0.5 MCLK
set_clock_transition 0.08 MCLK
set_clock_latency -source -max 4 MCLK
set_clock_latency -max 2 MCLK
```

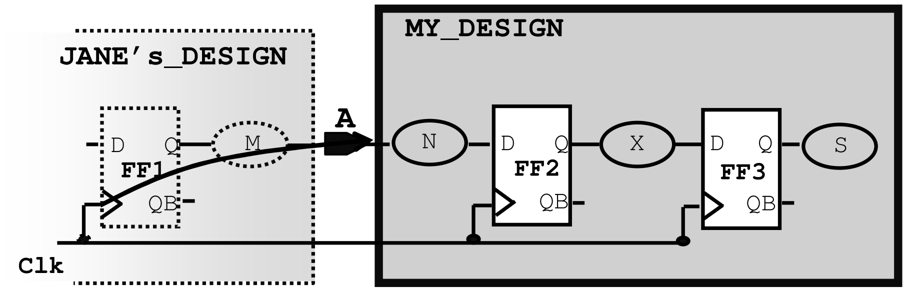
Post-CTS STA Constraints

```
reset_design
create_clock -p 5 -n MCLK Clk
set_clock_uncertainty 0.2 MCLK

set_clock_latency -source -max 4 MCLK
set_propagated_clock MCLK
```

4- 19

For post-CTS (Clock Tree Synthesis) static timing analysis `set_propagated_clock` forces the analyzer to calculate the ACTUAL clock tree skew, latency and transition times. The post-CTS constraints therefore do not include ideal transition and network latency commands. If the uncertainty number used during synthesis includes jitter and/or margin, these effects must still be included in the post-CTS analysis, along with the external source latency.

Constraining Input Paths



What additional information must <u>you</u> provide to constrain all the input paths (N) in your design for setup time?

4- 20

ANSWER: In addition to the clock definition, the user must supply the latest arrival time of the data at input port A, with respect to FF1's launching clock edge. In other words, an amount of time AFTER the external (Jane's) launching clock edge.

DC retrieves the setup time of FF2 from the technology library.

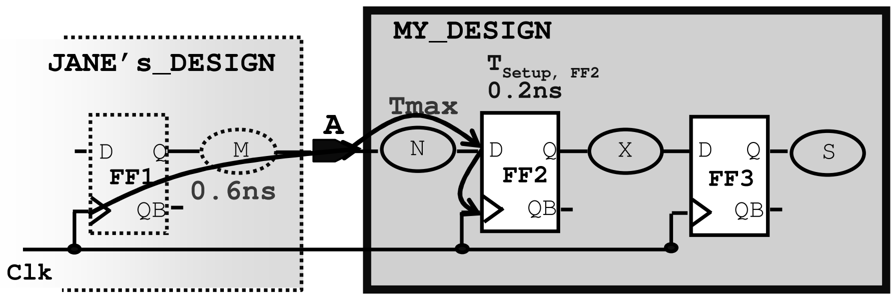
Constraining Input Paths: Example 1

Spec:

Latest Data Arrival Time at Port A, after Jane's launching clock edge = 0.6ns

mydesign.con

```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk [get_ports A]
```



What is the maximum delay Tmax for the input path N in MY_DESIGN? ____________________________

4-21

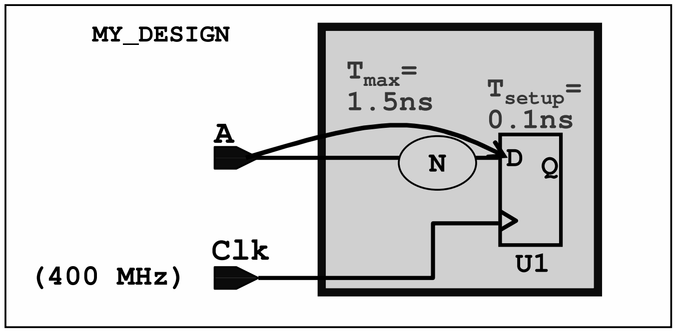
T input_logic, max = P – Input_Delay – Setup_Time (assuming zero clock skew)

Tmax = 2 - 0.6 - 0.2 = 1.2ns

Constraining Input Paths: Example 2

Spec:

Clock frequency = 400MHz. Maximum delay for path N = 1.5ns



How do you constrain MY_DESIGN for the indicated T_{max}?

```
create_clock ______________ [get_ports Clk]

set_input_delay _______________________________
```

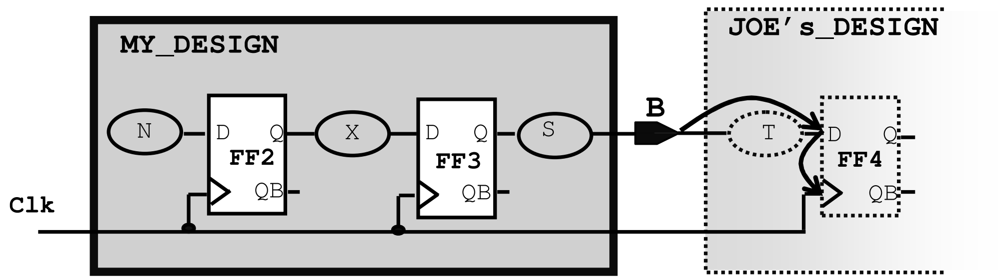
4- 22

```
create_clock -period 2.5 [get_ports Clk]
set_input_delay -max 0.9 -clock Clk [get_ports A]
```

(2.5 – 1.5 – 0.1 = 0.9ns)

Answer: 11.6

Constraining Output Paths



What additional information must you provide to constrain all the output paths (S) in your design for setup time?

4- 23

In addition to the clock definition, the user must supply the latest arrival time of the data at output port B, with respect to FF4's capturing clock edge, in other words, the maximum "setup time" at Joe's input, with respect to Joe's clock. This is an amount of time BEFORE the external capturing clock edge.

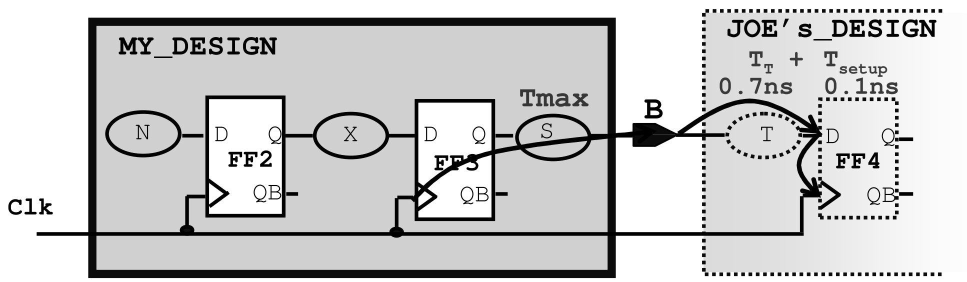
Constraining Output Paths : Example 1

Spec:

Latest Data Arrival Time at Port B, before Joe's capturing clock = 0.8ns

mydesign.con

```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk [get_ports A]
set_output_delay -max 0.8 -clock Clk [get_ports B]
```



What is the maximum delay Tmax for the output path through S in MY_DESIGN? ______________________

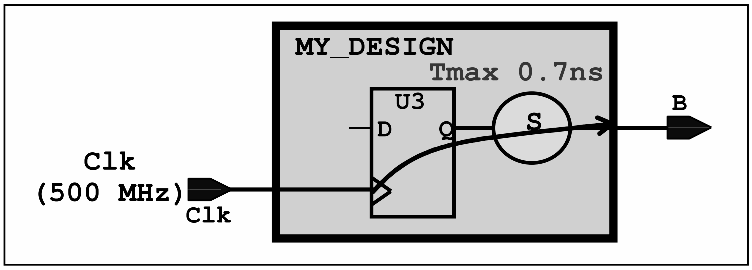
4-24

Toutput_logic, max = P – Output_Delay (assuming 0 clock skew)
Tmax = 2 – 0.8 = 1.2 ns

Constraining Output Paths : Example 2

Spec:

The maximum delay to Port B = 0.7ns



? How do you constrain MY_DESIGN for the indicated T_{max}?

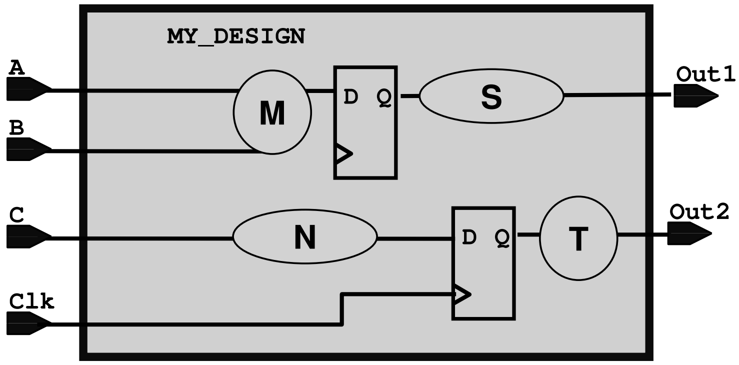
```
create_clock -period 2 [get_ports Clk]

set_output_delay ________________________________
```

4- 25

```
set_output_delay -max 1.3 -clock Clk [get_ports B]
```

Answer: 11.6

Multiple Inputs/Outputs - Same Constraints



To constrain all inputs the same, except for the clock port:

```
set_input_delay -max 0.5 -clock Clk \
  [remove_from_collection [all_inputs] [get_ports Clk]]
```

To constrain all outputs the same:

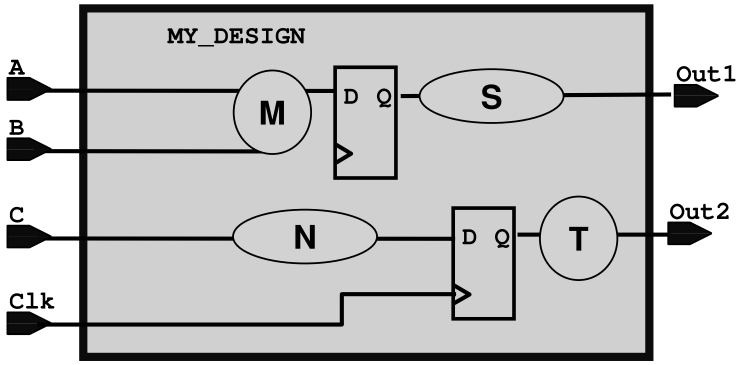
```
set_output_delay -max 1.1 -clock Clk [all_outputs]
```

4- 26

If you need to remove more than just one clock, use the following syntax:

```
remove_from_collection [all_inputs] [get_ports "Clk1 Clk2"]
```

Different Port Constraints



To constrain most ports the same, except for some:

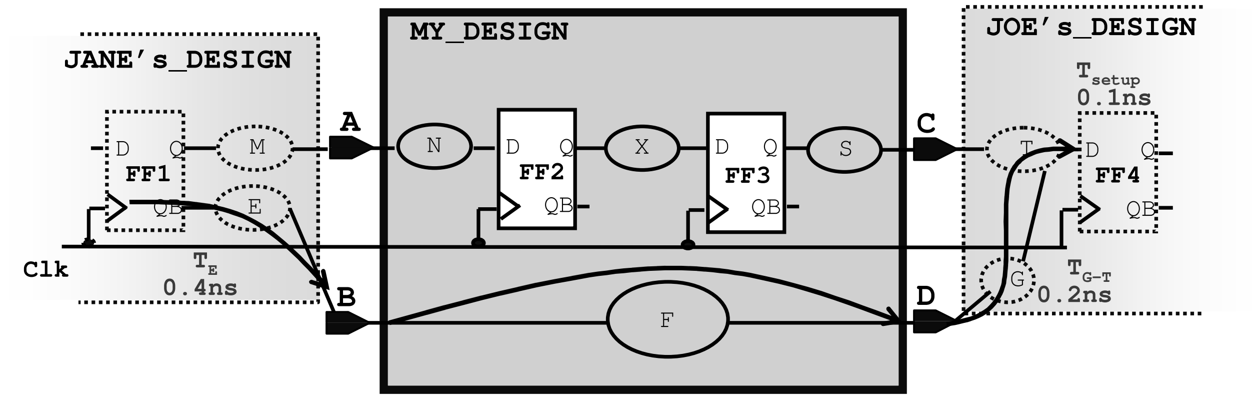
Overrides

```
set_input_delay -max 0.5 -clock Clk [all_inputs]
set_input_delay -max 0.8 -clock Clk [get_ports C]
remove_input_delay [get_ports Clk]
```

Another way to remove the constraint from the Clk port

4- 27

Exercise: Constraining Combinational Paths



How do you constrain the combinational path F?
What is the maximum delay through F?

```
create_clock -period 2 [get_ports Clk]

set_input_delay ________________________________

set_output_delay ________________________________
```

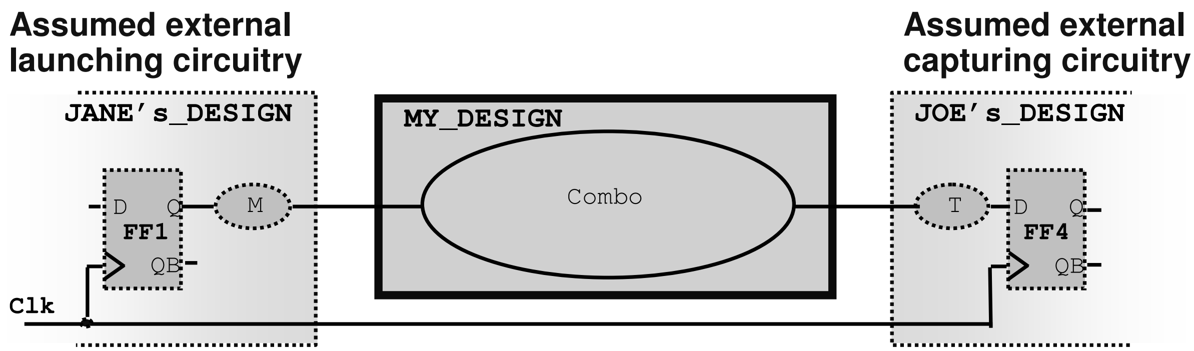
$T_{F,\,max}$ = ____________________

4-28

```
set_input_delay -clock Clk -max 0.4 [get_ports B]
set_output_delay -clock Clk -max 0.3 [get_ports D]
```

$T_{F,\,max} = 2 - 0.4 - 0.3 = 1.3\text{ns}$

Constraining a Purely Combinational Design



? What is different about this design?

? How do we constrain such a design?

4- 29

ANSWER: There is no clock input.

Answer: Use a Virtual Clock!

What is a virtual clock?

ANSWER:

- **A clock that is *not connected* to any port or pin within the current design**
- **Serves as a *reference* for input or output delays**
- **Creates a *clock object* with a *user-specified name* within Design Compiler's memory**

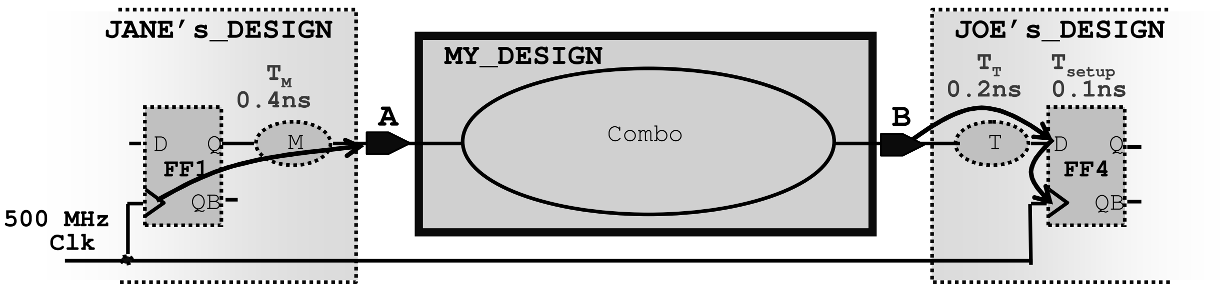
```
create_clock  -name VCLK  -period 2
```

Must be named

No source pin or port!

4-30

Exercise: Combinational Designs



How do you constrain the *Combo* path?
What is the maximum delay through *Combo*?

create_clock ________________________________

set_input_delay ________________________________

set_output_delay ________________________________

$T_{Combo,\ max}$ = ____________________

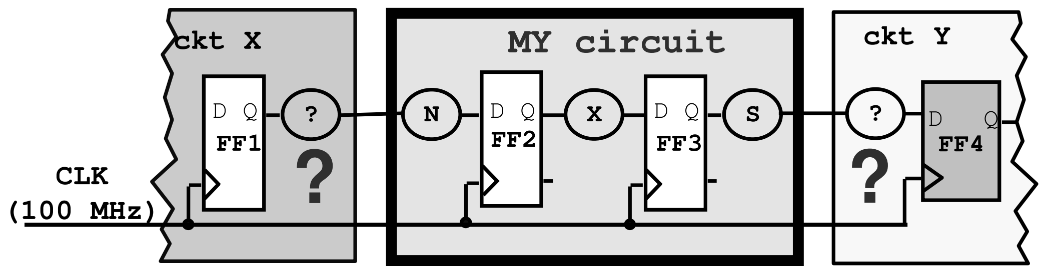
4-31

```
create_clock -name VClk -period 2
set_input_delay -clock VClk -max 0.4 [get_ports A]
set_output_delay -clock VClk -max 0.3 [get_ports B]
```

$T_{Combo,\ max} = 2 - 0.4 - 0.3 = 1.3\text{ns}$

Time Budgeting (1/2)

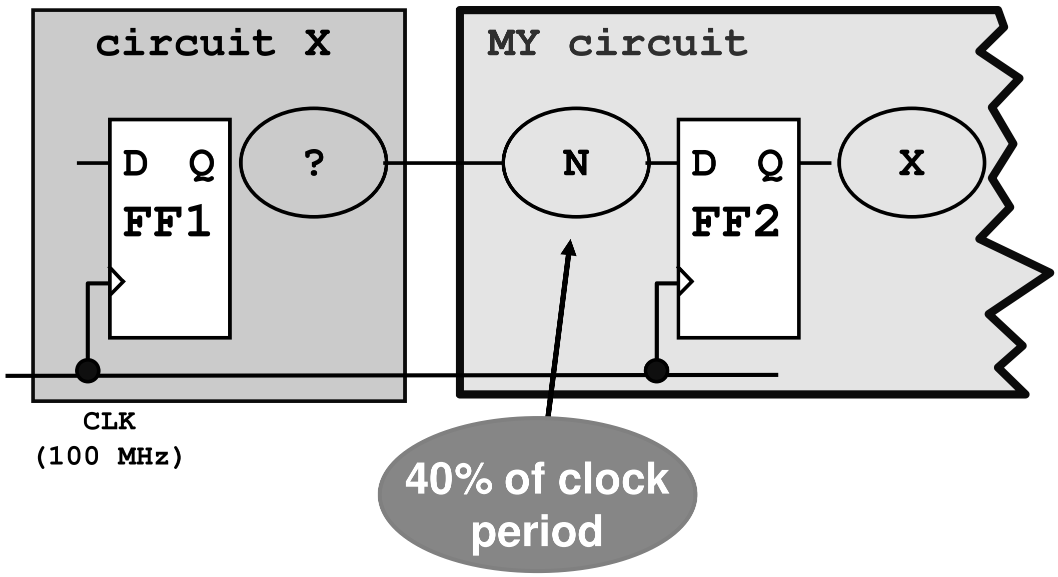
What if you do not know the delays on your inputs or the setup requirements of your outputs?



A: Create a Time Budget!

4-32

Time Budgeting (2/2)



Better to budget conservatively than to compile with paths unconstrained!

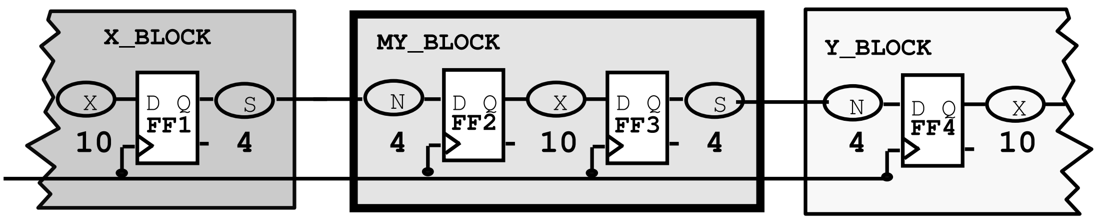
4- 33

The goal of this example is to tell Design Compiler to only use 40% of the clock period for the input logic cloud. If the designer for Circuit X does the same for the output logic cloud, there will be a 20% margin, including the delay of FF1 and the setup time of FF2.

Time Budgeting Example

timing_budget.tcl

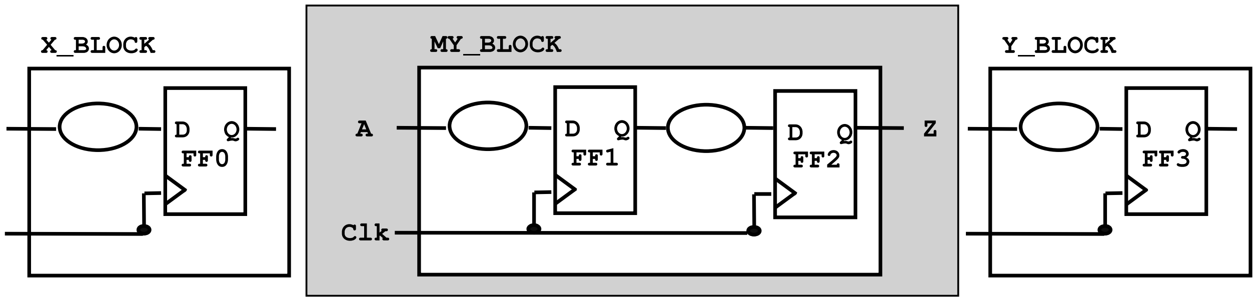
```
# A generic Time Budgeting script file
# for MY_BLOCK, X_BLOCK and Y_BLOCK
create_clock -period 10 [get_ports CLK]

set_input_delay -max 6 -clock CLK [all_inputs]
remove_input_delay [get_ports CLK]
set_output_delay -max 6 -clock CLK [all_outputs]
```



Would it be easier to specify a time budget if all outputs were registered?

4- 34

Registered Outputs



 # Assume every block has registered outputs, 10ns clock:

```
set clk_to_q_max 1.5; # Assume slowest register driving your input
set clk_to_q_min 0.9; # Assume fastest register driving your output
set all_in_ex_clk [remove_from_collection \
    [all_inputs] [get_ports Clk]]
set_input_delay  -max   $clk_to_q_max       -clock CLK $all_in_ex_clk
set_output_delay -max [expr 10 - $clk_to_q_min] -clock CLK [all_outputs]
```

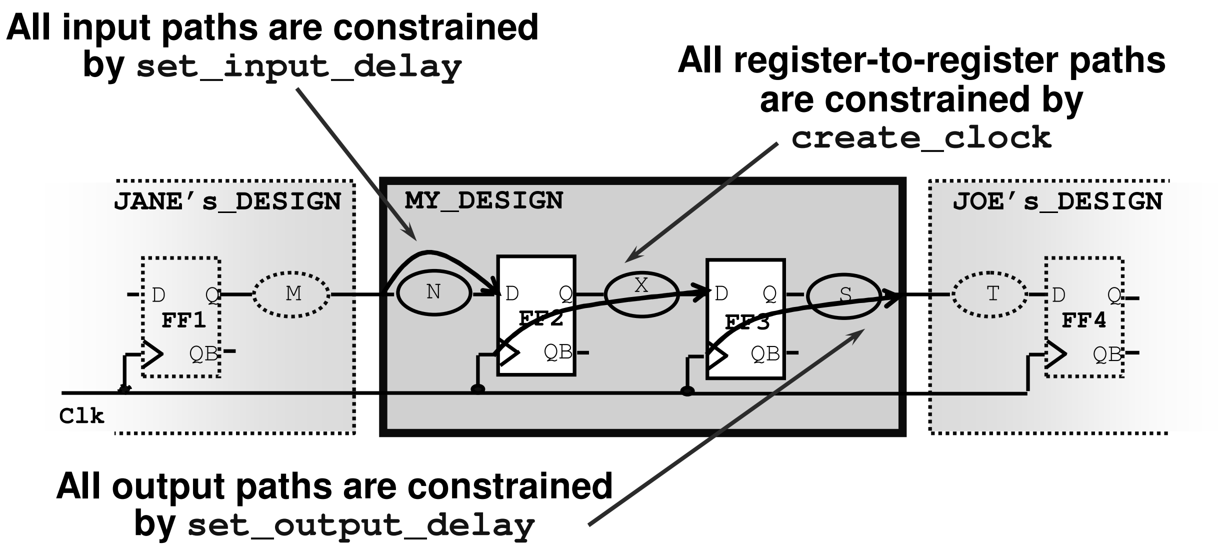
TCL: Arithmetic expression

4- 35

Additional examples using the TCL `expr` command:

```
set period 10.0; # Returns “10.0”
set freq [expr  1 / $period]; # Returns “0.1”
echo “Freg = ” [expr $freq * 1000] “MHz”; # Returns
                                          # “Freq = 100.0 MHz”

set_load [expr [load_of ssc_core_slow/and2a0/A] * 5] \
[all_outputs]; # Applies 5 and2a0 gate load to all outputs
```

Timing Constraint Summary



You specify how much time is used by <u>external</u> logic...

DC calculates how much time is left for the <u>internal</u> logic.

4- 36

Executing Commands Interactively

- **Commands can be typed interactively in DC-shell:**
  - OK for testing or debugging individual commands
  - Not efficient for “production work”

```
UNIX% dc_shell-t
dc_shell-xg-t> read_verilog {A.v B.v TOP.v}
dc_shell-xg-t> current_design MY_TOP
dc_shell-xg-t> link
dc_shell-xg-t> check_design
dc_shell-xg-t> set_max_area 245000
dc_shell-xg-t> create_clock -period 2 [get_ports Clk]
dc_shell-xg-t> set_input_delay -max 0.6 -clock Clk [all_inputs]
               ...
```

4- 37

Sourcing Constraints Files

- **For better efficiency, capture the constraints in a "constraints file", which can be executed interactively in DC-shell:**

```
dc_shell-xg-t> read_verilog {A.v B.v TOP.v}
dc_shell-xg-t> current_design MY_TOP
dc_shell-xg-t> link
dc_shell-xg-t> check_design
dc_shell-xg-t> source -echo -verbose TOP.con
...
```

TOP.con

```
set_max_area 245000
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk \ [all_inputs]
...
```

4- 38

A program is available for users to migrate from "old" dcsh to DC-Tcl. Will convert most commands in existing scripts to Tcl. Only goes from DCSH to DC-Tcl:

UNIX% `dc-transcript old_script.scr new_script.tcl`

Executing *Run Scripts* in "Batch Mode"

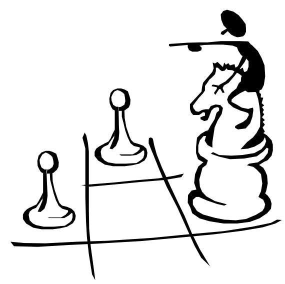
- **For maximum efficiency capture 'run commands' in a 'run script' and execute in batch mode**

```
UNIX% dc_shell-t -f RUN.tcl | tee -i run.log
```

  - Allows you to spend time on "other tasks" while DC is running
  - Do this only once you are certain the run script and constraints file are complete and correct

RUN.tcl

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
check_design
source -echo -verbose TOP.con
...
```



Other tasks

4- 39

The tee command displays the results on the screen while simultaneously writing them into the specified log file for later viewing.

The "-i" option tells the tee command to ignore interrupts. For example, if you enter <Ctrll-C> while in DC shell, the interrupt will be ignored by the tee command and passed on to the dc_shell command, which is what you intended. Otherwise, tee will process the <Ctrl-C> first.

Constraints File Recommendations (1 of 3)

Since constraints are saved in the *ddc* design format it is recommended to:

Erase all *constraints* from the current design before applying new constraints.

```
reset_design
set_max_area 245000
create_clock -period 2 [get_ports Clk]
...
```

When applying *multiple* constraint scripts, there should only be ONE `reset_design` command.

4- 40

Constraints File Recommendations (2 of 3)

Include comments in your scripts

```
# Comments in Tcl

# If you want to comment on the same line, be sure
# to use a semicolon before the comment:

create_clock -p 5 -n V_Clk; # This is a VIRTUAL clock
```

This semicolon is required!

Comment a line in a DC-Tcl script using the '#' character

4- 41

Constraints File Recommendations (3 of 3)

- **Use common extensions:**

  e.g. `RUN.tcl` **or** `DESIGN.con`

- **Avoid using <u>aliases</u> and abbreviating <u>commands</u>**
- **Avoid abbreviating command <u>options</u>:**

```
create_clock -period 5 [get_ports clk]
```

- **Avoid "snake scripts"**

4- 42

"Snake scripts" are scripts that call scripts, that call scripts: Very hard to debug.

Avoid sourcing scripts from your .synopsys_dc.setup file, since these scripts will be executed automatically every time you start the tool. This of course excludes scripts that only define procedures for later use.

Check the Syntax of Constraints

```
UNIX% dcprocheck TOP.con
```

```
UNIX% dcprocheck TOP.con
...

Unknown option 'create_clock -freq'
create_clock -freq 3.0 [get_ports clk]
                   ^
```

- *dcprocheck* **is a syntax-checking utility that is included with Design Compiler**
- **Available if you can launch DC – no additional user setup required**

4- 43

Check the Values/Options of Constraints

- **Verify port constraints (non-clock)**

```
report_port -verbose
```

```
dc_shell-xg-t> report_port -verbose
...
 Output Delay
                  Min               Max      Related  Fanout
Output Port   Rise     Fall     Rise     Fall  Clock      Load
-------------------------------------------------------------
EndOfInstrn   --       --       3.0      3.0      Clk      0.00
OUT_VALID     --       --       --       --       --       0.00
PSW[0]        --       --       --       --       --       0.00
...
```

- **Verify clock constraints – the commands show the:**

`report_clock`	-- clock waveform
`report_clock -skew`	-- clocktree specs

4- 44

Check for Missing/Inconsistent Constraints

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
link
check_design
source -echo -verbose TOP.con
report_port -verbose
report_clock
report_clock -skew
check_timing
 ...
```

Good practice:

`check_timing` after applying constraints

The `check_timing` command issues warnings for:

- **Missing endpoint constraints**
- **Missing, overlapping or multiple clocks**
- **Clock-gating signals that may interfere with the clock**
- **And more …**

4- 45

Redirect *Checks* and *Reports* to a File

It may be useful to redirect the output of run-script *checks* and *reports* to a separate file for later analysis

RUN.tcl

```
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
redirect -tee -file precompile.rpt {link}
redirect -append -tee -file precompile.rpt {check_design}
redirect -append -tee -file precompile.rpt {source -echo -ver TOP.con}
redirect -append -tee -file precompile.rpt {report_port -verbose}
redirect -append -tee -file precompile.rpt {report_clock}
redirect -append -tee -file precompile.rpt {report_clock -skew}
redirect -append -tee -file precompile.rpt {check_timing}
 ...
```

4- 46

Need Help with Commands and Variables?

- **Commands:**

```
help *clock;                    # Lists all matching
                                # commands

help -verbose create_clock; # Lists command options
create_clock -help;         # Same

man create_clock;           # Complete 'man page'
```

- **Variables:**

```
printvar *_library;   # Lists all matching variables
                      # and corresponding values

echo $target_library; # Echoes the variable value

man target_library;   # Complete 'man page'
```

4-47

```
dc_shell-xg-t> help *clock
 clock                     # Builtin
 create_clock              # create_clock
 create_test_clock         # create_test_clock
 remove_clock              # remove_clock
 remove_propagated_clock   # remove_propagated_clock
 report_clock              # report_clock
 set_propagated_clock      # set_propagated_clock

dc_shell-xg-t> help -verbose create_clock
 create_clock  # create_clock
   [-name clock_name]      (name for the clock)
   [-period period_value]  (period of the clock)
   [-waveform edge_list]   (alternating rise, fall times for
                            1 period)
   [port_pin_list]        (list of ports and/or pins)
```

Summary: Commands Covered (1 of 2)

TOP.con

```
# Area and Timing Constraints
#

reset_design;  # Good practice step

set_max_area 245000
create_clock -period 2 [get_ports Clk]
create_clock -period 3.5 -name V_Clk; # VIRTUAL clock
set_clock_uncertainty -setup 0.14 [get_clocks Clk]
set_clock_latency -source -max 0.3 [get_clocks Clk]
set_clock_latency -max 0.12 [get_clocks Clk]
set_clock_transition 0.08 [get_clocks Clk]
set_input_delay -max 0.6 -clock Clk [all_inputs]
set_input_delay -max 0.5 -clock V_Clk [get_ports "A C F"]
set_output_delay -max 0.8 -clock Clk [all_outputs]
set_output_delay -max 1.1 -clock V_Clk [get_ports "OUT2 OUT7]
```

4- 48

Summary: Commands Covered (2 of 2)

RUN.tcl

```
# Run script
#
read_verilog {A.v B.v TOP.v}
current_design MY_TOP
redirect -tee -file precompile.rpt {link}
redirect -append -tee -file precompile.rpt {check_design}
redirect -append -tee -file precompile.rpt {source -echo -ver TOP.con}
redirect -append -tee -file precompile.rpt {report_port -verbose}
redirect -append -tee -file precompile.rpt {report_clock}
redirect -append -tee -file precompile.rpt {report_clock -skew}
redirect -append -tee -file precompile.rpt {check_timing}
 ...
```

```
UNIX% dc_shell-t -f RUN.tcl | tee -i my.log
```

```
help *clock
help -verbose create_clock
create_clock -help
printvar *_library
man target_library
```

4- 49

Summary: Unit Objectives

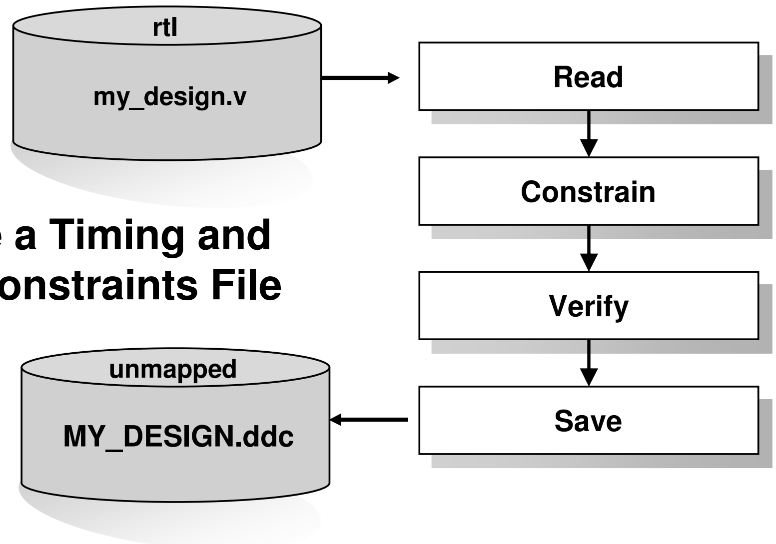
You should now be able to:

- **Constrain a design for area**
- **Constrain a design for <u>setup</u> timing, under the following conditions:**
  - The design's specs are given or known
  - Can be block- or chip-level design
  - Single clock, single cycle environment
  - Default design scenario – minimal command options
- **Create and execute a constraints file**

4- 50

Lab 4: Timing and Area Constraints

80 minutes



Create a Timing and Area Constraints File

4- 51

This page was intentionally left blank.

Agenda

DAY 2

5	**Partitioning for Synthesis**	
6	**Environmental Attributes**	
7	**Compile Commands**	
8	**Timing Analysis**	
9	**More Constraint Considerations**	

Synopsys 10-I-011-SSG-013

5- 1

Unit Objectives

After completing this unit, you should be able to:

- List two effects of partitioning a circuit through combinational logic
- State the main guideline for partitioning for synthesis
- State how partitions are created in HDL code
- List two DC commands for modifying partitions

5- 2

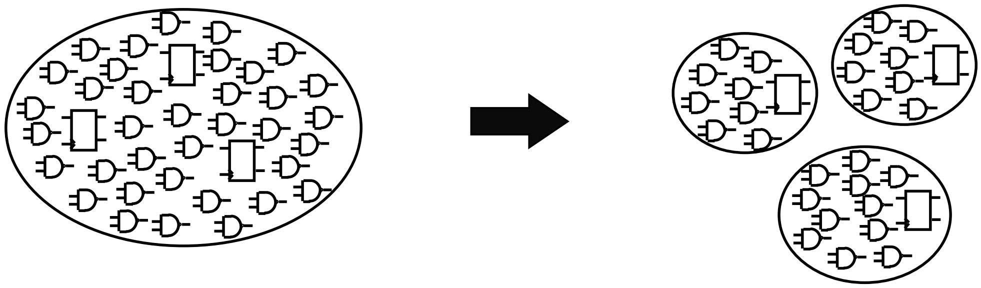
Commands Covered in this Unit

```
# Automatic ungrouping by DC

compile_ultra # Auto-ungrouping enabled by default
compile -auto_ungroup area | delay
compile -ungroup_all

set_dont_touch

# Manual re-partioning by the user

group -design NEW_DES -cell U23 {U2 U3}
ungroup -start_level 2 U23
```

5- 3

What Is Partitioning? Why Partition?

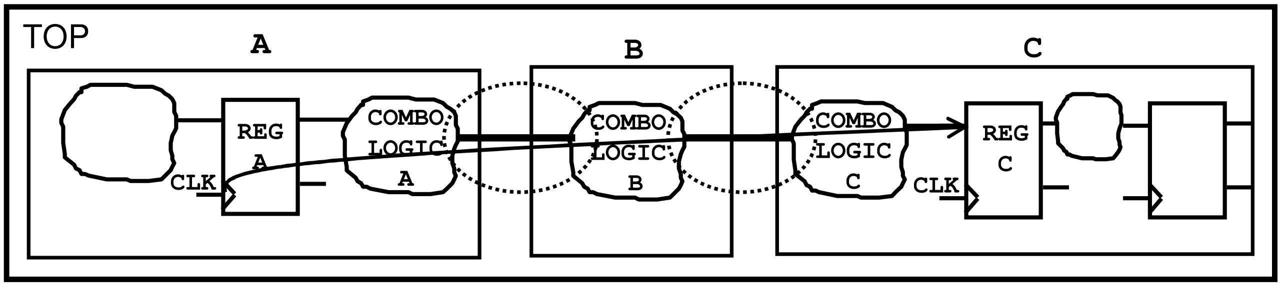
***Partitioning*: Dividing large designs into smaller parts**



- **Partitioning is driven by many needs:**
  - Separate distinct functions
  - Achieve workable size and complexity
  - Manage project in team environment
  - Design Reuse
  - Meet physical constraints, and more …

5- 4

Poor Partitioning



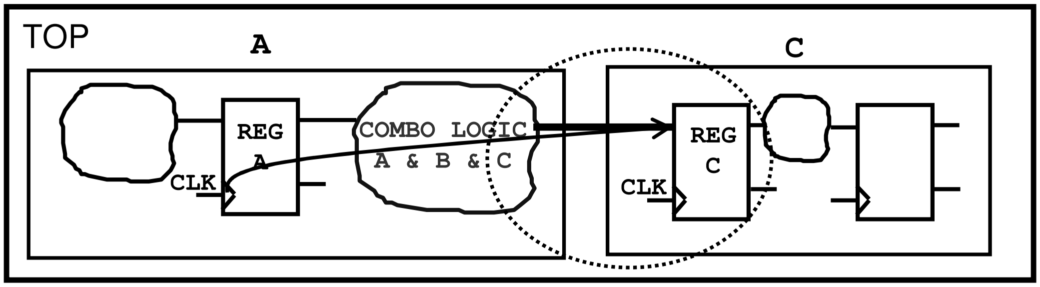
How does this partitioning affect synthesis?

- **Design Compiler must preserve block pin definitions**
  - Logic optimization – e.g. merging of combinational logic - does not occur across block boundaries
- **Path from REG A to REG C may be larger and slower than necessary → Poorly partitioned!**

5- 5

Better Partitioning

Guideline: Do not 'slice' through combinational paths

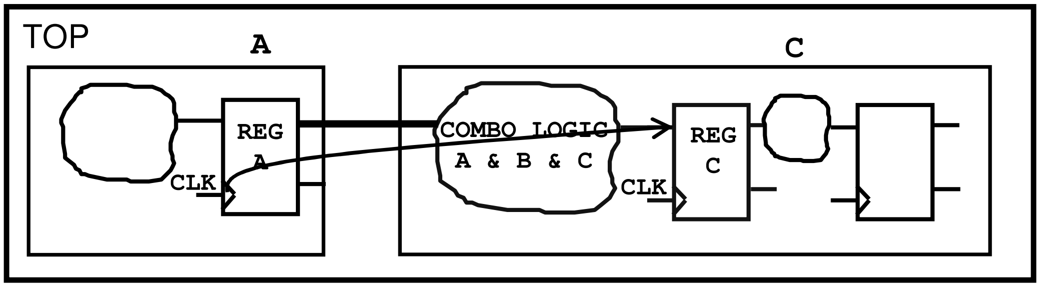


- **Related combinational logic is grouped into one block:**
  - No hierarchy separates combinational functions A, B, and C
- **Combinational optimization techniques can now be fully exploited → Faster and smaller combo logic!**
- **However, no *sequential optimization* possible at REG C**

5- 6

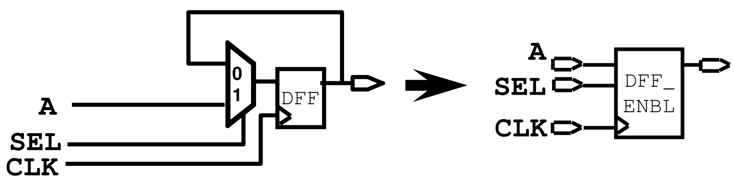
Best Partitioning

Guideline: Do not 'slice' at register inputs – rather at outputs



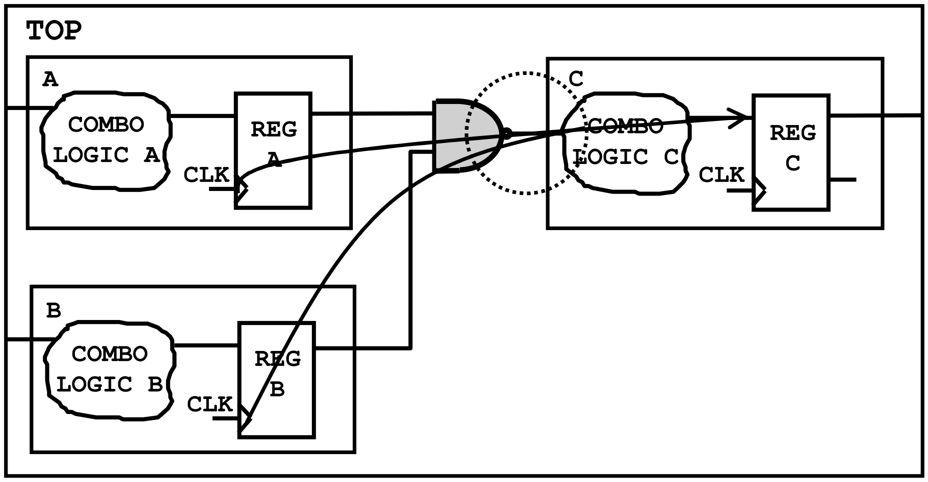
- **Sequential optimization may now integrate some of the combinational logic into a more complex Flip-Flop**

(JK, Toggle, Muxed Flip-Flops …)



5- 7

Corollary Guideline: Avoid Glue Logic



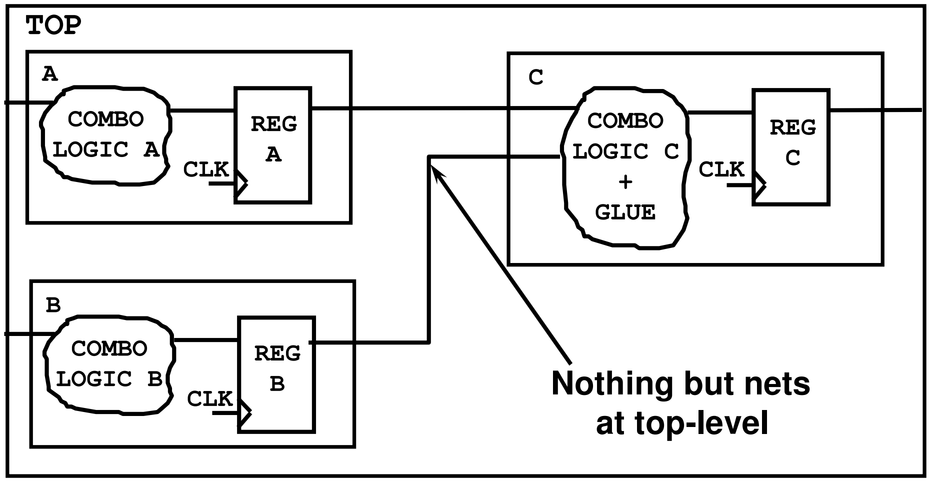
Poor Partitioning

Issues:

- **No combinational optimization between glue logic and combo logic C**
- **If the top-level blocks A, B and C are large and will be synthesized separately (*middle-up* compile strategy), an additional compile is needed at top-level**

5- 8

Remove Glue Logic Between Blocks

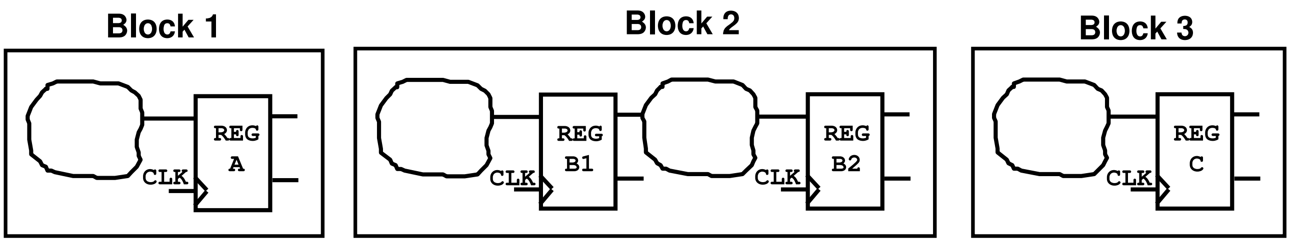


Good Partitioning

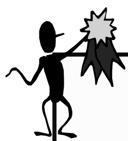
- The glue logic can now be optimized with other logic
- Top-level design is only a structural netlist, it does not need to be compiled

5- 9

In Summary …



Best Partitioning



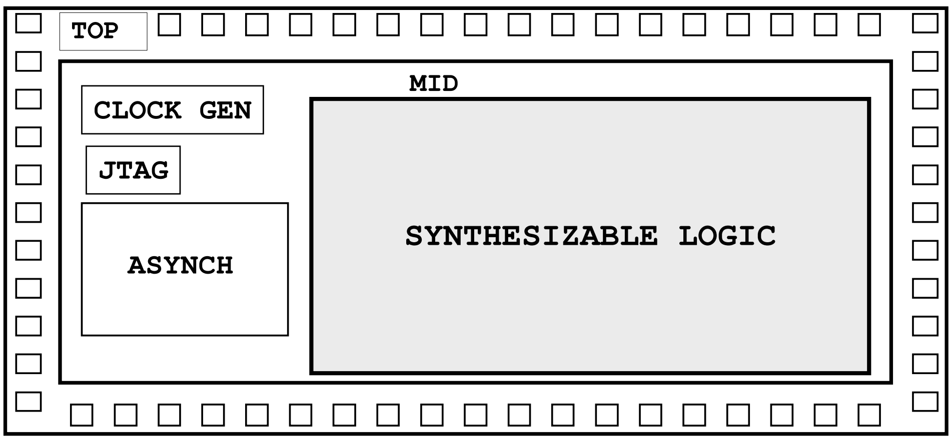
If partitioning is necessary, try to place the hierarchy boundaries at register outputs

- **Allows efficient combinational and sequential optimization along timing paths**
- **Simplifies timing constraints for sub-block synthesis:**
  - The arrival times of the inputs to each block is a register Clk→Q delay
  - Input logic path delay has almost the entire clock period

5- 10

Additional Guidelines

- **Design Compiler has no inherent design size limit**
- **Compile as large a design as possible for best QoR**
  - Size is limited only by available *memory* resources and run time
- **Separate the synthesizable logic from the non-synthesizable logic – apply *dont_touch* for top-down compile**



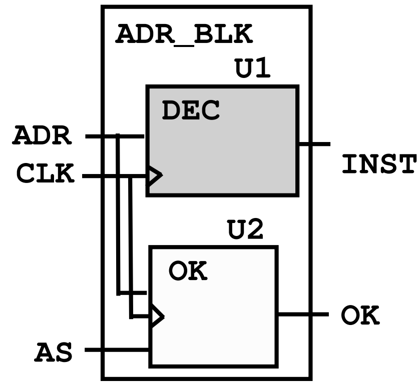
5- 11

Most designs contain some circuitry that is pre-designed and not intended to be synthesized, for example: Clock generators (PLL, divide-by); JTAG boundary scan logic; non-synthesizable Asynchronous circuitry; I/O pad cells. It makes things a lot easier if this circuitry is defined as its own hierarchy ("hard macros" or hard IP) and instantiated in the design, separate from the logic intended for synthesis. If you intend to perform a top-down compile at the chip level (recommended if the design fits in memory) then let DC know not to optimize these blocks with `set_dont_touch true [instantiated_blocks]`, and compile from the TOP design.

Partitioning Within the HDL Description

```
entity ADR_BLK is... end;
architecture STR of ADR_BLK is
 U1:DEC port map(ADR, CLK, INST);
 U2:OK  port map(ADR,CLK,AS,OK);
end STR;
```

```
module ADR_BLK (...
 DEC U1(ADR,CLK,INST);
 OK  U2(ADR,CLK,AS,OK);
endmodule
```



- **`entity` and `module` statements define hierarchical blocks:**
  - Instantiation of an entity or module creates a new level of hierarchy
- **Inference of Arithmetic Circuits (+, -, *, ..) can create a new level of hierarchy**
- **Process and Always statements do not create hierarchy**

5- 12

Partitioning in Design Compiler

- **Ideally, your RTL-level design follows the "good partitioning" guidelines – if so, you are done!**
- **What if your RTL design is poorly partitioned?**
  - If possible, re-partition in RTL (recommended)[1]
  - Otherwise, re-partition using Design Compiler
- **Partitions can be manipulated in two ways:**
  - Automatic
    - Synthesis re-partitions during *compile*
  - Manual
    - User re-partitions prior to *compile*

5- 13

[1] It is usually better for simulation purposes if your RTL code matches the hierarchy of your synthesized gate-level netlist. This allows you to re-use any simulation test-benches that were designed for RTL to simulate the gate-level netlist too. If the hierarchy is modified by DC, the simulation test-bench may need to be modified too.

From Design Compiler's perspective it makes absolutely no difference where, when or how the design is partitioned: You get the same results if the exact same good partitioning is achieved in RTL, or manually using DC commands prior to *compile*, or using auto-ungrouping during *compile*.

Automatic Partitioning

- **During synthesis, *auto-ungrouping* can automatically make "smart" re-partitioning choices [1]:**

```
compile_ultra; # auto-ungrouping enabled
               # by default

compile -auto_ungroup area|delay
```

  - Auto-ungrouping controlled through variables (Unit 11)
  - To report designs auto-ungrouped during synthesis use `report_auto_ungroup`

- **Or, ungroup the entire hierarchy [2]**

```
compile -ungroup_all
```

? What if you don't have Ultra and you can't ungroup_all?

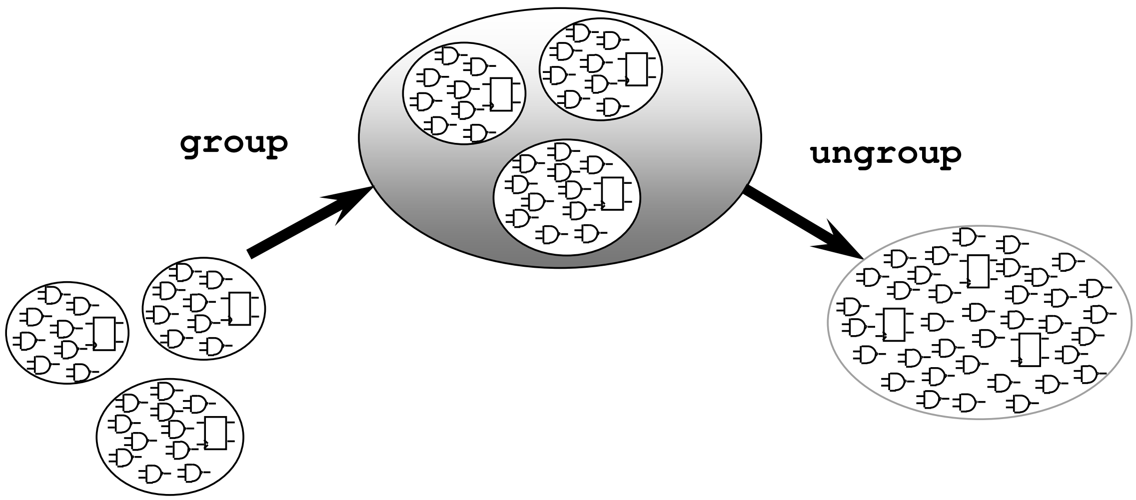
5- 14

[1] Requires an Ultra license
[2] Designs marked with a "dont_touch" attribute (`set_dont_touch`) are not compiled or ungrouped.

Manual Partitioning

Answer: Manually partition prior to compile

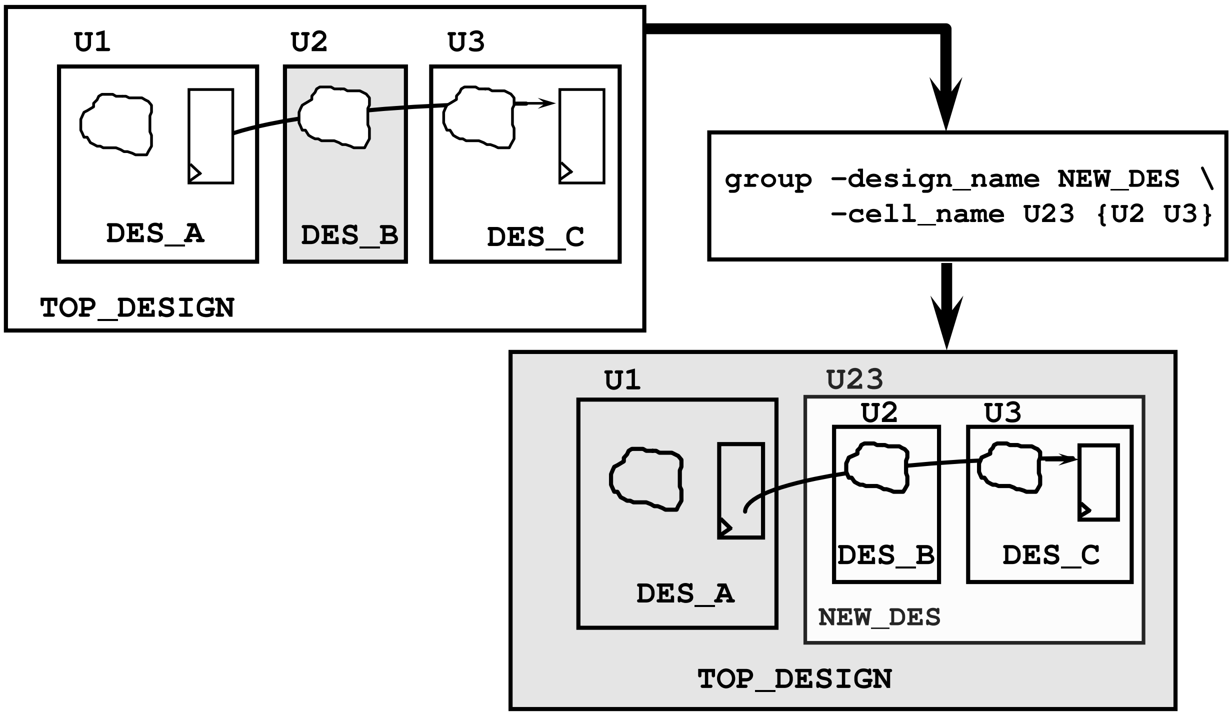
The `group` and `ungroup` commands modify the partitions in a design.



5- 15

The `group` Command

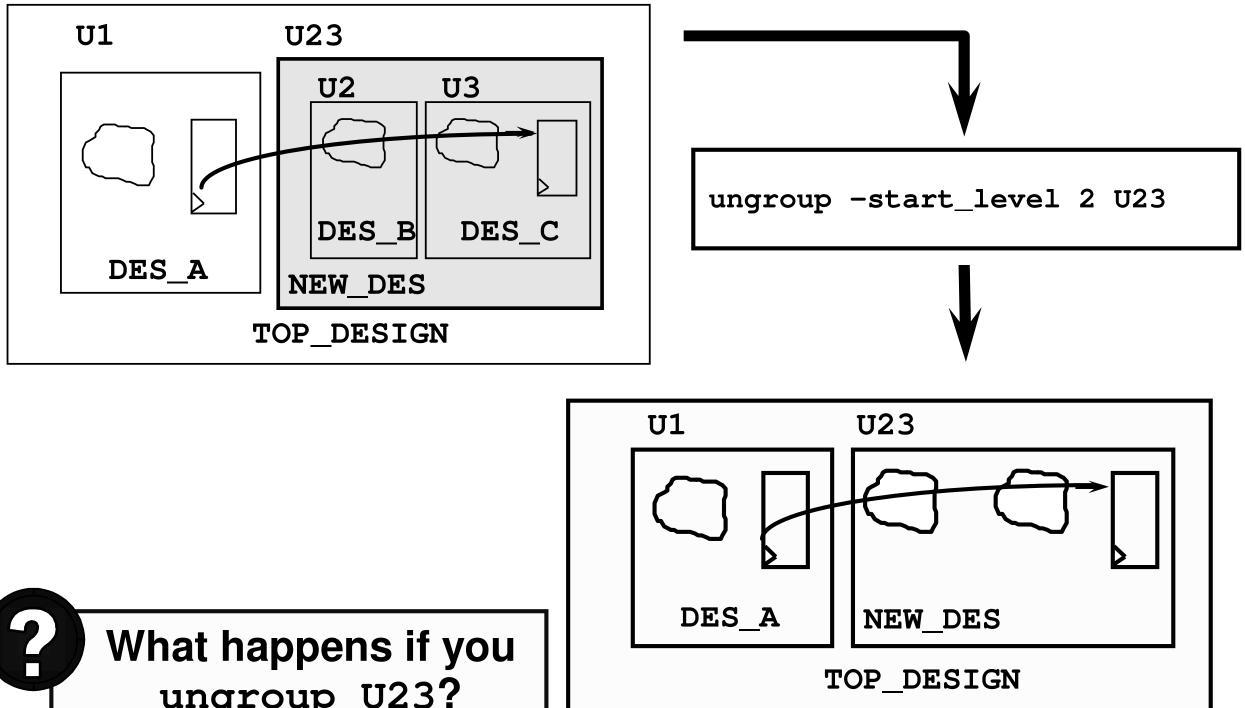
`group` creates a new hierarchical block



5- 16

The ungroup Command

ungroup removes either one or all levels of hierarchy



What happens if you ungroup U23?

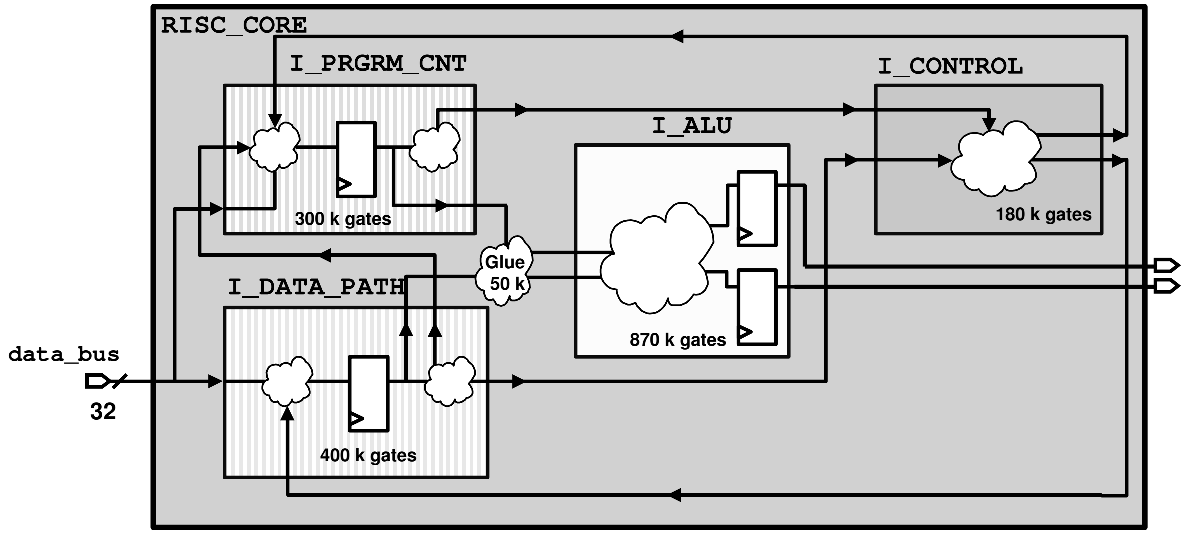
5- 17

To ungroup all hierarchy in the **current_design**.

```
ungroup -all -flatten
```

```
dc_shell-xg-t> ungroup -help
Usage: ungroup    # ungroup hierarchy
        [-all]                  (ungroup all cells)
        [-prefix <prefix>]      (prefix to use in naming cells)
        [-flatten]              (expand all levels of hierarchy)
        [-simple_names]         (use simple, non-hierarchical names)
        [-small <n>]            (ungroup all small hierarchy:
                                 Value >= 1)
        [-force]                (ungroup dont_touched cells as well)
        [-soft]                 (remove group_name attribute)
        [-start_level <n>]      (flatten cells from level:
                                 Value >= 1)
        [cell_list]             (list of cells to be ungrouped)
```

Exercise: Poor Partitioning



What's wrong with this design's partitioning?

How would you improve the partitioning of this design for synthesis?

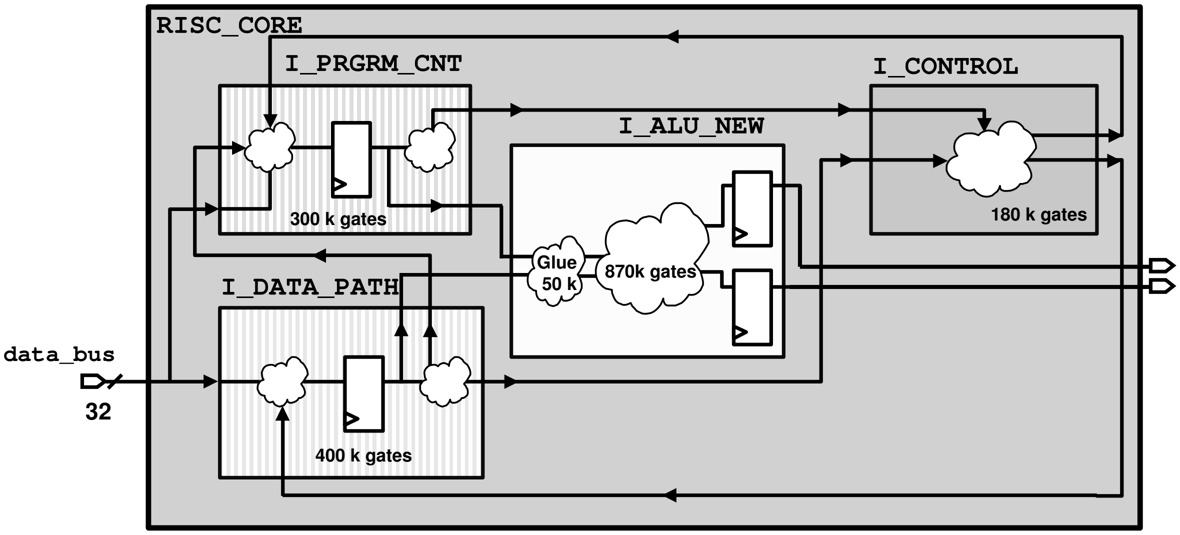
5- 18

If the RISC_CORE design above is NOT too large (hardware has enough memory and run time is acceptable) it is recommended to perform a "top-down" compile at the RISC_CORE level. All you need is the top-level constraints – the tool does the rest (deriving constraints for the sub-blocks and compiling them bottom-up). The term "top-down compile" is misleading. It represents what happens from the user's perspective, not from the tool's perspective. The user reads in the entire design hierarchy, sets the current_design to the top-level design, applies top-level constraints, and invokes a compile command at the top level. From the user's view, everything is done at the top level, and is then "propagated downward. DC can not actually compile the hierarchy top-down – it has to compile bottom-up. It must also derive constraints for these sub-blocks, which a "top-down compile" will do automatically.

If the RISC_CORE design above is too large to compile top-down (let's say you don't want to wait for the entire compile to finish before seeing result), YOU will need to compile it using a "bottom-up" approach. You are basically doing "manually" what a top-down compile does automatically. This will allow you to see synthesis results at the sub-block level, in less time.

In either case, the current partitioning is poor and may result is less-than-optimal synthesis results (timing and area). Furthermore, the "bottom-up" approach requires you to: derive accurate constraints for the 4 sub-blocks (difficult due to poor partitioning); compile each of the 4 block separately using a top-down compile; finally, place a "don't_touch" attribute on each of the sub-blocks, and, once they meet their individual constraints, perform a top-level compile just to get the glue logic synthesized. Even though the glue logic is relatively small, the compile time may not necessarily be fast, since the entire design is in memory, and DC must still time through the surrounding blocks.
How can you improve the partitioning of this design to help either compile approaches? →

Exercise: Move Glue Logic into Sub-block



current_design ________________

set GLUE_CELLS ________________

group ________________

ungroup ________________

What are the advantages of this?

5- 19

```
current_design RISC_CORE
set GLUE_CELLS [remove_from_collection [get_cells *] \
    [get_cells I_*]]
group -design ALU_NEW -cell I_ALU_NEW "$GLUE_CELLS I_ALU"
ungroup -start_level 2 I_ALU_NEW
```

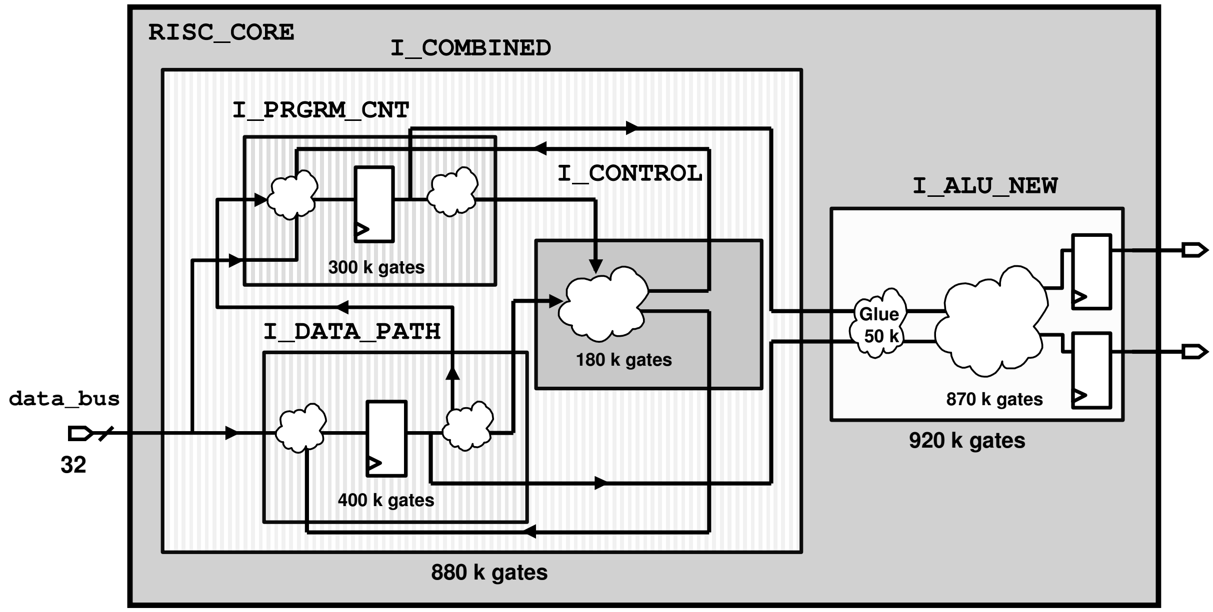
Advantages:

1. The glue logic and the ALU combo logic can now be optimized jointly, resulting in a potentially smaller/faster ALU_NEW.

For a bottom-up compile approach:

2. The inputs of ALU_NEW are now registers, which makes it easier to derive constraints for this block.
3. It is no longer necessary to perform a top-level compile just for the glue logic. Once the sub-blocks have been compiled you simple read in and link the top-level design, and verify that the constraints are being met at the top level.

Exercise: Combine Other Sub-blocks




```
...
group ____________________________________________
```

What are the advantages of this?

5- 20

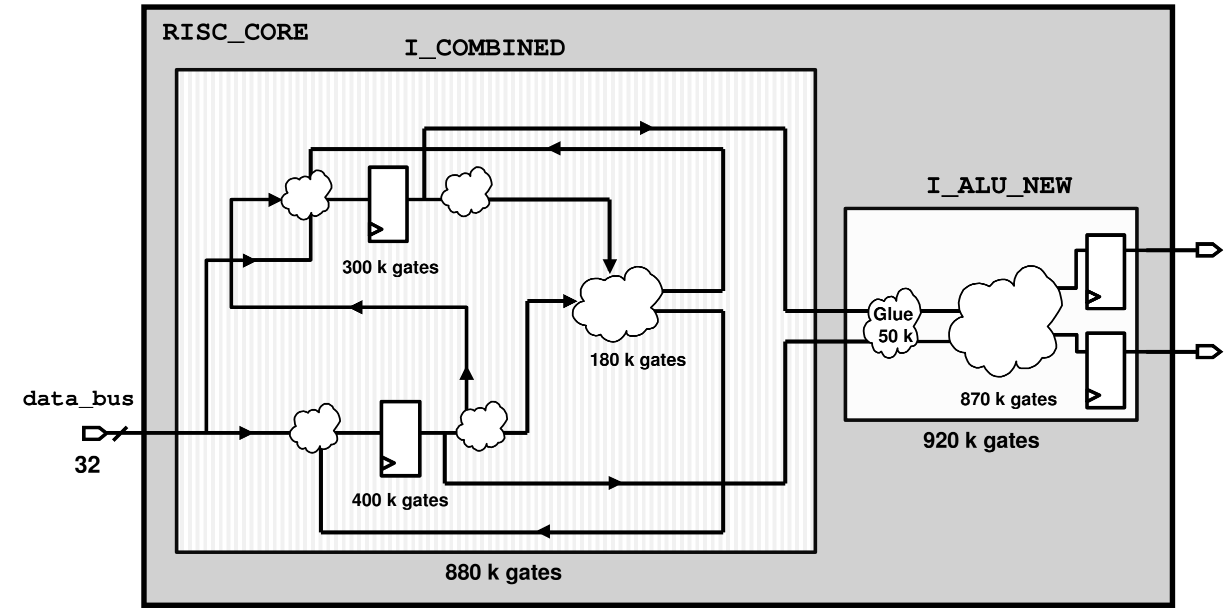
```
group -design COMBINED -cell I_COMBINED "I_C* I_D* I_P*"
```

Advantages:

1. For a top-down compile of RISC_CORE – no difference.

   For a bootom-up approach:
2. There are now only two sub-designs that need to be constrained and compiled, instead of four. The "COMBINED" block can now be compiled "top-down". This is sometimes referred to as a "middle-down" approach.
3. Both sub-designs are roughly the same size, so you compile times and memory requirements are significantly less than compiling RISC_CORE, yet balanced. Having as large a block as possible maximizes DC's efficiency and QoR.
4. Both top-level blocks have registered outputs and connect to the RISC_CORE level, so it is much simpler to derive their constraints.

Exercise: Ungroup 2nd Level Blocks




```
...
ungroup ________________________________________________
```

? What's the advantage of this?

5- 21

```
ungroup -start_level 2 I_COMBINED
```

Advantage:

For both top-down or bottom-up, there is no longer any hierarchy separating the combinational logic between the DATA_PATH, PRGRM_CNT and CONTROL blocks. DC can now optimize these combo clouds more effectively, to produce smaller and faster logic.

Partitioning Strategies for Synthesis

- Do not separate combinational logic across hierarchical boundaries
- Place hierarchy boundaries at register outputs
- Avoid glue logic at the top level
- Separate non-synthesizable and synthesizable logic

5- 22

Answer for the question on previous page:
The command '`ungroup {U23}`' will create two instances with the names `U23/U2` and `U23/U3` respectively.

Partitioning for Synthesis: Summary

What do you gain by "partitioning for synthesis"?

- Better results -- smaller and faster designs
- Easier synthesis process -- simplified constraints and scripts
- Faster compiles -- quicker turnaround

5- 23

Summary: Commands Covered

```
# Automatic ungrouping by DC

compile_ultra # Auto-ungrouping enabled by default
compile -auto_ungroup area | delay
compile -ungroup_all

set_dont_touch

# Manual re-partioning by the user

group -design NEW_DES -cell U23 {U2 U3}
ungroup -start_level 2 U23
```

5- 24

Summary: Unit Objectives

You should now be able to:

- **List two effects of partitioning a circuit through combinational logic**
- **State the main guideline for partitioning for synthesis**
- **State how partitions are created in HDL code**
- **List two DC commands for modifying partitions**

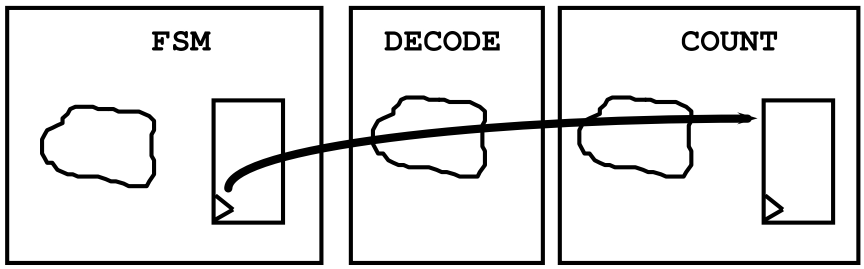
5- 25

Lab 5: Partitioning for Synthesis

30 minutes

`group / ungroup`

Design Problem: Timing Violation



Change Partitioning for Better Synthesis Results

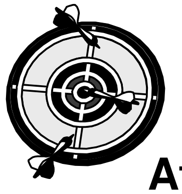
5- 26

Agenda

DAY 2

5	Partitioning for Synthesis
6	Environmental Attributes
7	Compile Commands
8	Timing Analysis
9	More Constraint Considerations

Unit Objectives

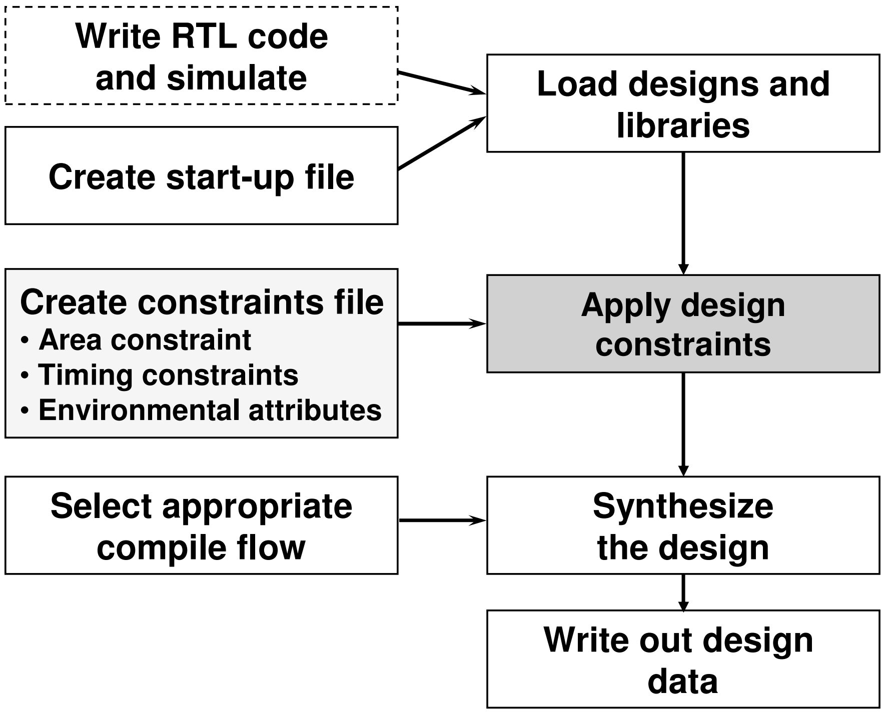


After completing this unit, you should be able to:

- **Apply environmental attributes to model the timing effects of:**
  - Input drivers and transition times
  - Capacitive output loads
  - PVT operating conditions or ‘corners’
  - Interconnect parasitic RCs

6- 2

RTL Synthesis Flow



6- 3

Commands To Be Covered (1 of 2)

TOP.con

```
# Area and Timing constraints go here
...
...
set_max_capacitance $MAX_INPUT_LOAD $all_in_ex_clk
set_load [expr $MAX_INPUT_LOAD * 3] [all_outputs]
set_load 0.080  [get_ports OUT23]
set_load [load_of slow_proc/NAND2_3/A] [get_ports OUT42]
set_input_transition 0.12 [remove_from_collection \
                          [all_inputs][get_ports B]]
set_driving_cell -lib_cell FD1 -pin Q [get_ports B]
set_operating_conditions -max WCCOM
set_wire_load_model -name 1.6MGates
```

6- 4

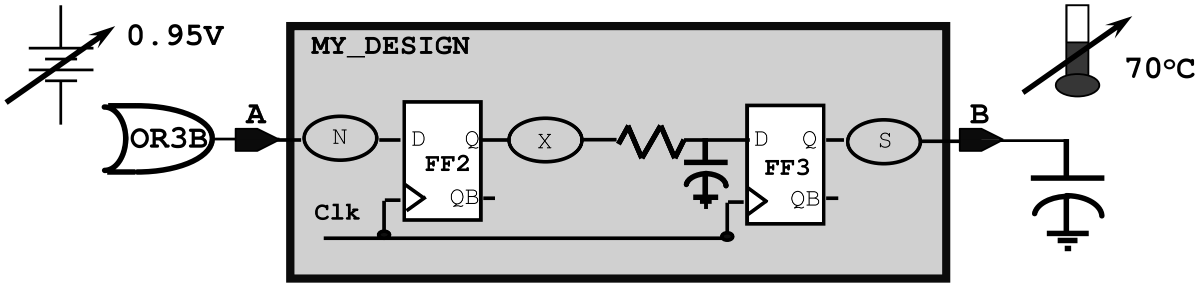
Commands To Be Covered (2 of 2)

MAX_CAP.tcl

```
# Algorithm to find largest max_capacitance in library and
# apply that value as a conservative output load

set LIB_NAME ssc_core_slow
set MAX_CAP 0
set OUTPUT_PINS [get_lib_pins $LIB_NAME/*/* -filter
"direction == 2"]
foreach_in_collection pin $OUTPUT_PINS {
   set NEW_CAP [get_attribute $pin max_capacitance]
   if {$NEW_CAP > $MAX_CAP} {
      set MAX_CAP $NEW_CAP
   }
}
set_load $MAX_CAP [all_outputs]
```

6- 5

Factors Affecting Timing



mydesign.con

```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk [get_ports A]
set_output_delay -max 0.8 -clock Clk [get_ports B]
```

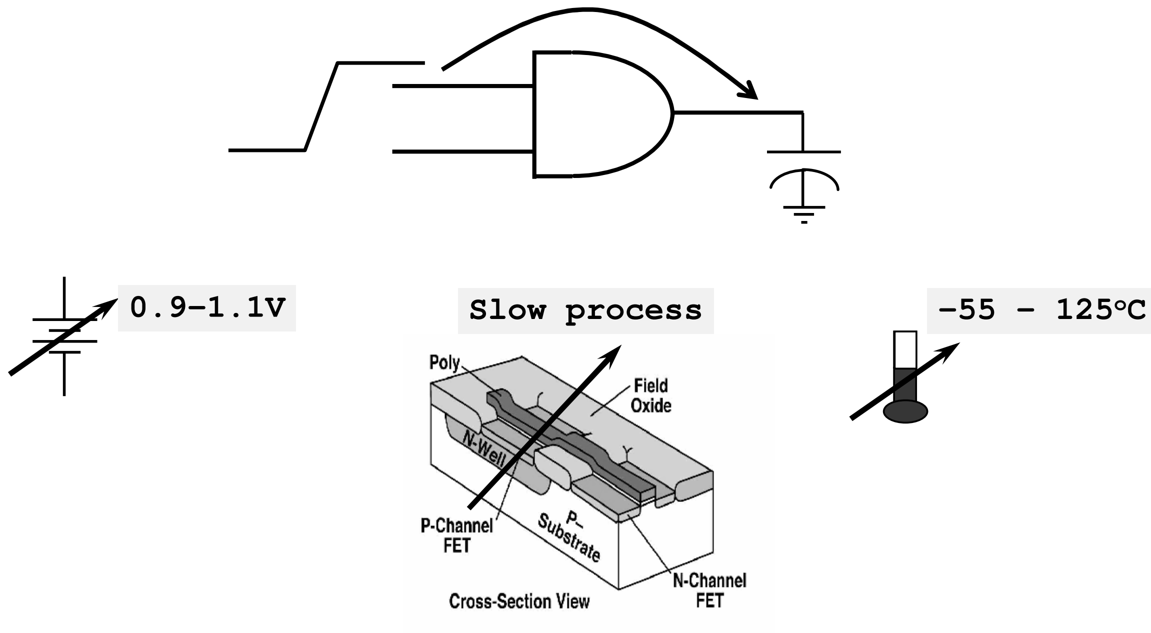
The above constraints are required, but not sufficient for DC to accurately model and optimize all logic path delays.
Need to take into account: Input drivers/transition times, Output loading, PVT corners and parasitic RCs

6- 6

The constraints determine how much time is available for reg-to-reg, input and output paths, but they do not describe under what conditions these delays must be met. Capacitive loading on outputs, transition times on inputs, process/voltage/temperature conditions, as well as interconnect parasitic RCs all affect the path delays, and must therefore be accurately modeled.

By default, DC assumes ideal (zero) loads and ideal (zero) transition times on input and output ports, ideal interconnect parasitics (Rnet, Cnet = 0), and no PVT scaling. This is a very optimistic environment.

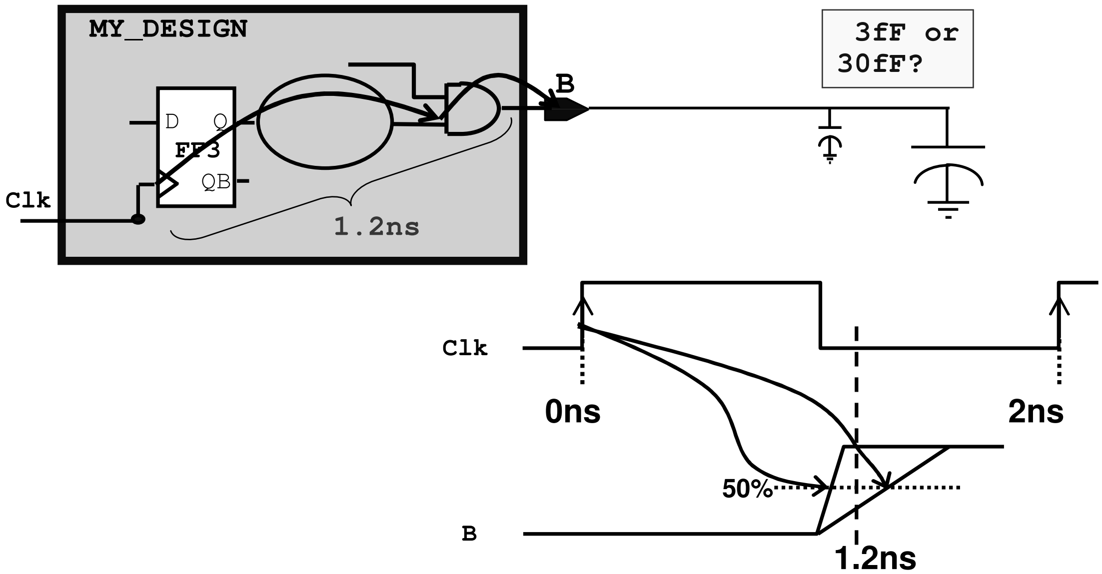
Cell Delay Dependencies



Cell delays are a function of input transition time, output capacitive loading, and PVT conditions

6- 7

Effect of Output Capacitive Load



Capacitive loading on an output port affects the transition time, and thereby the cell delay, of the output driver.

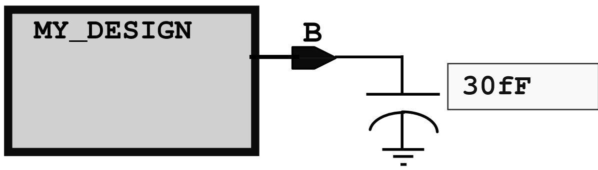
By default DC assumes zero capacitive loading on outputs. It is therefore important to accurately model capacitive loading on all outputs.

6- 8

Modeling Output Capacitive Load: Example 1

Spec:

Chip-level: Maximum capacitive load on output port B = 30fF



Unit of capacitance is 1pF in this example. Defined in the technology library.

mydesign.con

```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk [get_ports A]
set_output_delay -max 0.8 -clock Clk [get_ports B]
set_load [expr 30.0/1000] [get_ports B]
```

TCL: Arithmetic expression

What if a specific capacitance value is not known, at a block-level output port for example?

6- 9

Modeling Output Capacitive Load: Example 2

Spec:

Block-level: Maximum load on output port B = 1 "AN2" gate load, or = 3 "inv1a0" gates

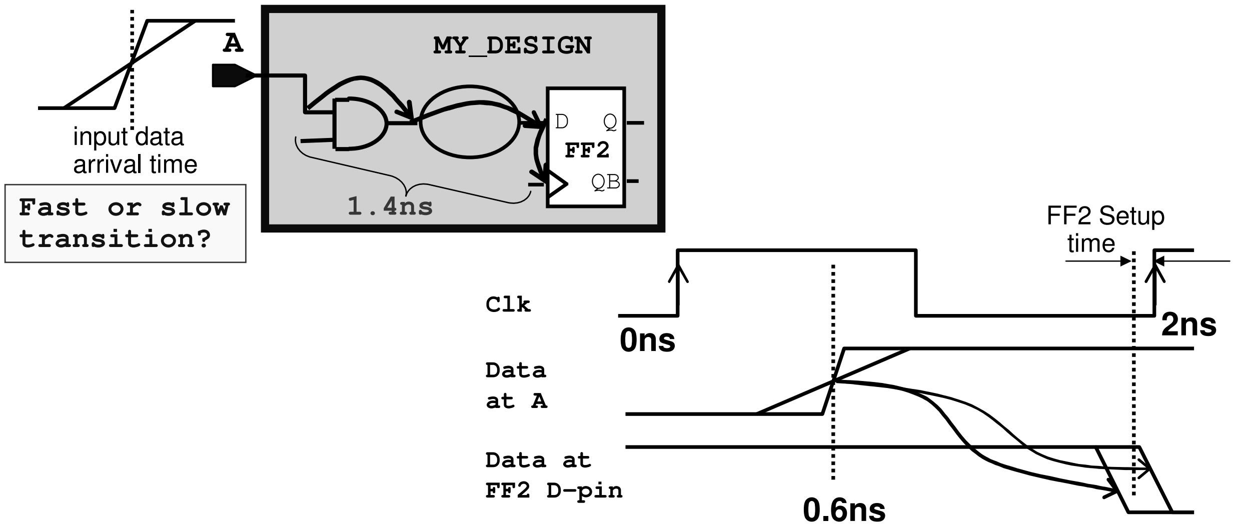
Use `load_of` *`lib/cell/pin`* to place the load of a gate from the technology library on the port:


```
set_load [load_of my_lib/AN2/A] [get_ports OUT1]
```

```
set_load [expr [load_of my_lib/inv1a0/A] * 3] \
  [get_ports OUT1]
```

6- 10

Effect of Input Transition Time



Rise and fall transition times on an input port affect the cell delay of the input gate.

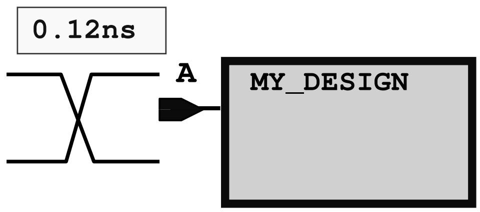
By default DC assumes zero transition times on inputs. It is therefore important to accurately model transition times on all inputs.

6- 11

Modeling Input Transition: Example 1

Spec:

Chip-level: Maximum rise/fall input transition on input port A = 0.12ns



mydesign.con

```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk [get_ports A]
set_output_delay -max 0.8 -clock Clk [get_ports B]
set_load [expr 30.0/1000] [get_ports B]
set_input_transition 0.12 [get_ports A]
```

? **What if a specific transition time value is not known, at a block-level input port for example?**

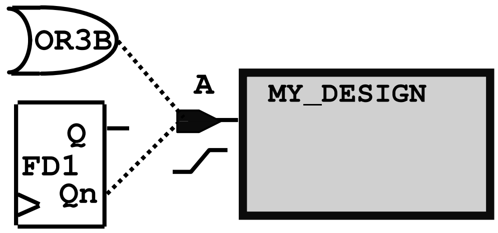
6- 12

Modeling Input Transition: Example 2

Spec:

Block-level: Driving cell on input port A = OR3B gate, or

= Qn pin of FD1 flip-flop



mydesign.con

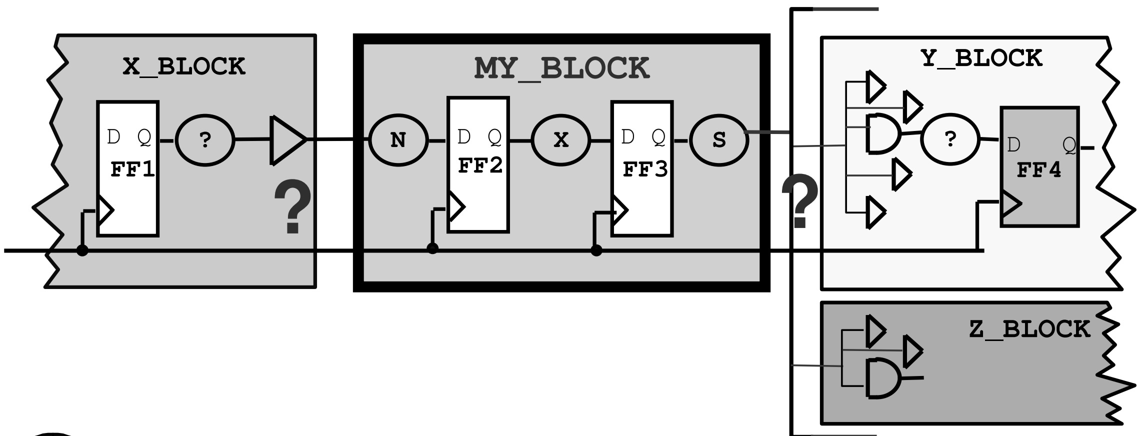
```
create_clock -period 10 [get_ports Clk]
set_input_delay -max 3 -clock Clk [get_ports A]
set_output_delay -max 4 -clock Clk [get_ports B]
set_load [expr 30.0/1000] [get_ports B]
set_driving_cell -lib_cell OR3B [get_ports A]
             or
set_driving_cell -lib_cell FD1 -pin Qn [get_ports A]
```

If no pin is given, DC will use first output pin listed in the library cell definition!

6- 13

Load Budgeting (1/2)

What if, prior to compiling, the cells driving your inputs, and the loads on your outputs are not known?



A: Create a Load Budget!

6- 14

Load Budgeting (2/2)

- **Creating a load budget:**
  - Assume a weak cell driving the inputs (to be conservative)
  - Limit the input capacitance of each input port
  - Estimate the number of other major blocks your outputs may have to drive

? **How do you limit the input capacitance of an input port?**

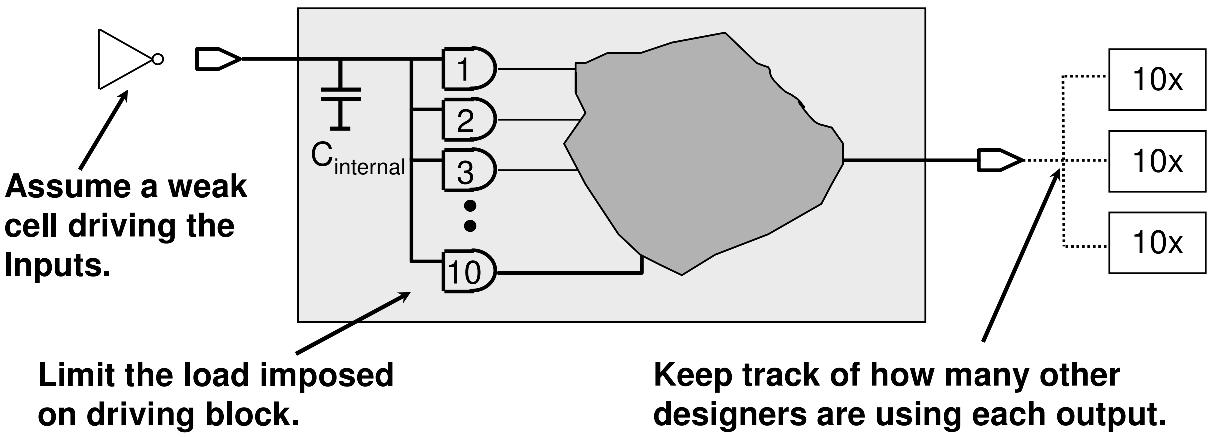
A: Place restrictive design rules on your input ports.

6- 15

Load Budget Example (1/2)

Assumptions:

1. The maximum fanout capacitance of any block's input port is limited to the equivalent of 10 **"`and2a1`"** gates
2. Output ports can drive a maximum of 3 other blocks
3. The driving gate of every output is the cell **`inv1a1`**



6- 16

Load Budget Example (2/2)

```
current_design myblock
link
reset_design
source timing_budget.tcl
set all_in_ex_clk [remove_from_collection \
    [all_inputs] [get_ports Clk]]

# Assume a weak driving buffer on the inputs
set_driving_cell -lib_cell inv1a1 $all_in_ex_clk

# Limit the input load1
set MAX_INPUT_LOAD [expr \
        [load_of ssc_core_slow/and2a1/A] * 10]
set_max_capacitance $MAX_INPUT_LOAD $all_in_ex_clk

# Model the max possible load on the outputs, assuming
# outputs will only be tied to 3 subsequent blocks1
set_load [expr $MAX_INPUT_LOAD * 3] [all_outputs]
```

Script from Unit 4!

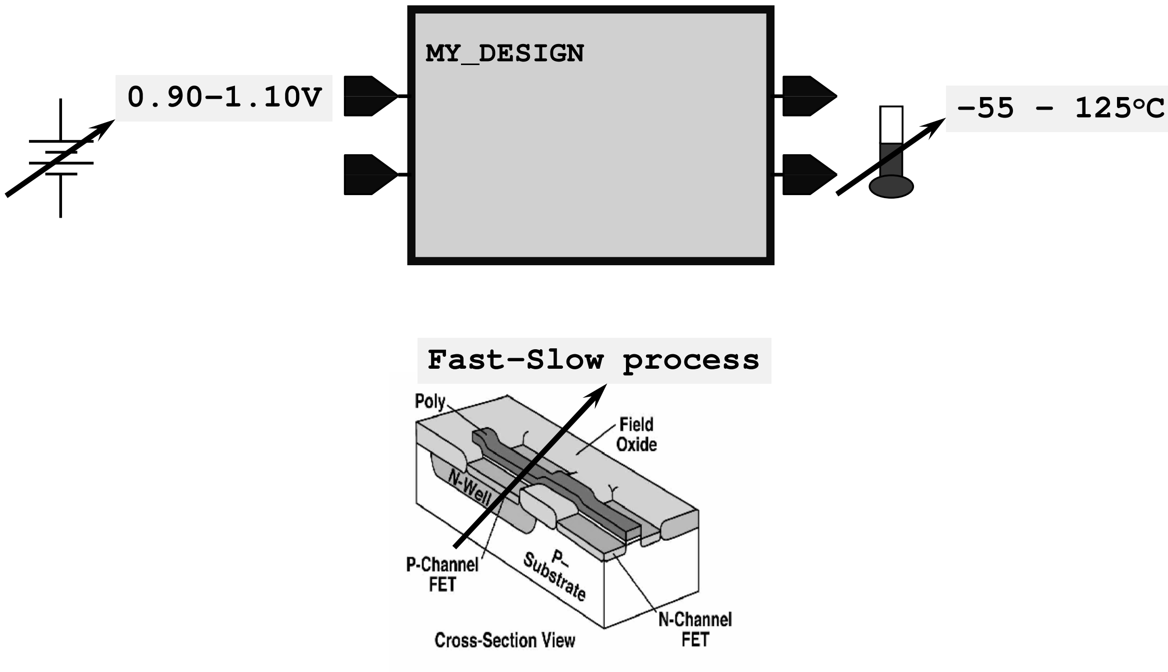
6- 17

The set_max_capacitance command is a "design rule constraint" (DRC) which limits the fanout capacitance (pin + wire) of a port. By default, DRCs have higher priority than timing and area constraints, so it is very important to not over-constrain user-defined DRCs.

[1] To determine the conservative load of your block's outputs, the above example requires that you limit the maximum fanout capacitance of all block's input ports to an arbitrarily large value. You must also estimate (arbitrarily) the maximum number of blocks that any of your output ports can drive. Alternatively, the following method is a way of applying a conservative output load which is not arbitrary, but is instead determined by your library cell characteristics. The method assumes that each output pin of your library cells has a vendor-defined max_capacitance attribute and value. It determines the largest max_capacitance value in the entire library. Since, by default, no output port is able to drive a larger load than this maximum value without violating a DRC, you set the load of each of your outputs to this maximum value.

```
set LIB_NAME ssc_core_slow
set MAX_CAP 0
set OUTPUT_PINS [get_lib_pins $LIB_NAME/*/* -filter "direction == 2"]
foreach_in_collection pin $OUTPUT_PINS {
  set NEW_CAP [get_attribute $pin max_capacitance]
  if {$NEW_CAP > $MAX_CAP} {
    set MAX_CAP $NEW_CAP
  }
}
set_load $MAX_CAP [all_outputs]
```

Modeling PVT Effects



6- 18

PVT Effects: Three Library Scenarios

Your library supplier will provide you with either:

1. **One “nominal” technology library file, which includes “PVT corner” derating models (least common)**
2. **Multiple technology library files, each characterized for a different “PVT corner” – no derating models included (most common)**
3. **Multiple technology library files, each including derating models**

CHECK WITH YOUR LIBRARY SUPPLIER!!

6- 19

Examples of scenario #3:

a. Each file is characterized for different PROCESS (P) corners, e.g. slow, nominal and fast, but for NOMINAL V and T. Operating conditions models are then included in each file to derate V and T for different corners, e.g high and low commercial, industrial and/or military voltage and temperatures.

b. The library files are characterized for slow and fast COMMERCIAL PVT only, and additional operating conditions are included in each library to further derate the PVT for INDUSTRIAL and/or MILITARY conditions.

Case 1: One Lib File with Operating Cond’s

- **Use `list_lib` to find the library name**

```
Library       File              Path
-----------------------------------------
nom_90nm     Xvendor_90.db    /project/lib
```

- **Use `report_lib` *`nom_90nm`* to list the derating models, or *operating conditions*:**

```
Operating Conditions:
    Name           Library       Process    Temp     Volt
    ----------------------------------------------------

    lib default                    1.00     25.00    1.00
    WCCOM           nom_90nm       1.50     70.00    0.95
    WCMIL           nom_90nm       1.50    125.00    0.90
    BCCOM           nom_90nm       0.60      0.00    1.05
    BCMIL           nom_90nm       0.60    -55.00    1.10
```

- **Select the operating condition with:**

```
set_operating_conditions –max “WCCOM”
```

6- 20

To display the default operating conditions, if specified, use the following command:

`get_attribute` *`my_lib`* **`default_operating_conditions`**

my_lib = the name of your technology *library* (not the library *file* name).

Case 2: Multiple Lib Files, No Op. Cond’s

Example:

Your library supplier provides you with the following files:

- *Xvendor_90nm_nominal.db* (characterized for nominal PVT)
- *Xvendor_90nm_wccom.db* (characterized for slow comm’l PVT)
- *Xvendor_90nm_bccom.db* (characterized for fast comm’l PVT)

In your .synopsys_dc.setup file, enter:

.synopsys_dc.setup

```
set target_library Xvendor_90nm_wccom.db
set link_library "* Xvendor_90nm_wccom.db"
```

NO NEED TO SET AN OPERATING CONDITION!!

6- 21

Case 3: Multiple Files with Operating Cond's

Example: You are provided with the following files:

- *Xvendor_90nm_slow_proc.db* (slow process, nominal VT)
- *Xvendor_90nm_fast_proc.db* (fast process, nominal VT)

```
Operating Conditions:
    Name            Library        Process     Temp     Volt
    ---------------------------------------------------------
    lib default                     1.50      25.00     1.00
    WCCOM            slow_proc      1.50      70.00     0.95
    WCMIL            slow_proc      1.50     125.00     0.90
```

Define your library and operating conditions as:

.synopsys_dc.setup

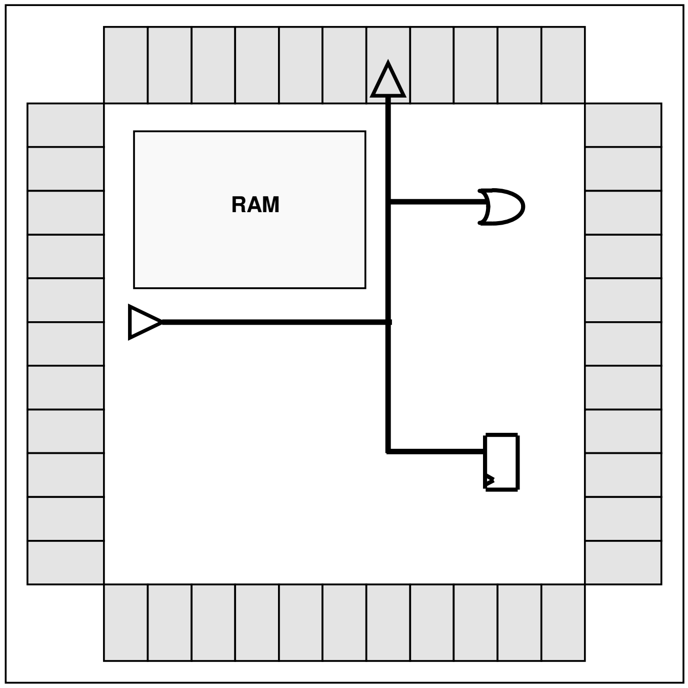
```
set target_library vendorX90nm_slow_proc.db
set link_library "* vendorX90nm_slow_proc.db"
```

mydesign.con

```
set_operating_conditions -max "WCCOM"
```

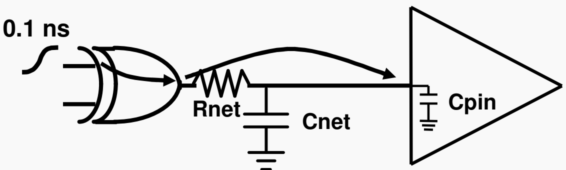
6- 22

Modeling Interconnect or Net Parasitics



6- 23

Path Delays are Based on Cell + Net Delays



Cell Delay = f**(Input Transition Time,** $C_{net} + C_{pin}$**)**

Net Delay = f**(**R_{net}**,** $C_{net} + C_{pin}$**)**

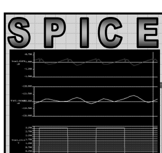
Cell and net delays are both a function of parasitic RCs

? **Prior to layout, how can the net parasitic RCs be estimated?**

6- 24

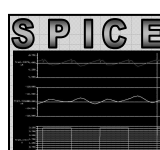
Cell delay is calculated using non-linear delay models, which are stored in the ‘LM’ view of each cell. NLDM is highly accurate as it is derived from SPICE characterizations. The delay is a function of the input transition time of the cell (TInput) [also called slew], the driving strength of the cell (RCell), the wire capacitance (CNet) and the pin capacitance of the receivers (CPin). A slow input transition time will slow the rate at which the cell’s transistors can change state (from “on” to “off”), as well as a large output load (Cnet + Cpin), thereby increasing the “delay” of the logic gate.

There is another NLDM table in the library to calculate output transition. Output transition of a cell becomes the input transition of the next cell down the chain.



SPICE	Output Load (pF)			
Input Trans (ns)	.005	.05	.10	.15
0.0	.1	.15	.2	.25
0.5	.15	. 23	.3	.38
1.0	.25	.4	.55	.75

Cell Delay (ns)

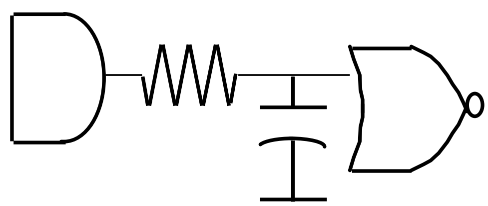


SPICE	Output Load (pF)			
Input Trans (ns)	.005	. 05	.10	.15
0.00	0.10	0.20	0.37	0.60
0.50	0.18	0.30	0.49	0.80
1.00	0.25	0.40	0.62	1.00

Output Transition (ns)

Modeling Net RCs with Wire Load Models

- **A wire load model calculates one parasitic R and one C for each net, based on the net's fanout:**
  - Models for various design sizes are supplied by your vendor
  - R/C values are average estimates based on data extracted from similar designs which were fabricated using this process



6- 25

Wire Load Model Examples

What does this mean???

```
Name            : 140000
Location        : 90nm_com
Resistance      : 0.000331
Capacitance     : 8.6e-05
Area            : 0.1
Slope           : 93.7215
Fanout   Length
-------------------------
     1    14.15
     2    32.31
     3    52.48
     4    74.91
     .
     .
    20   952.16
```

kΩ/unit length

pF/unit length

area/unit length

Extrapolation slope

```
Name            : 8000000
Location        : 90nm_com
Resistance      : 0.000331
Capacitance     : 8.6e-05
Area            : 0.1
Slope           : 334.957
Fanout   Length
-------------------------
     1     24.58
     2     58.28
     3     98.54
     4    146.54
     .
     .
    20  2946.37
```

Ask your library provider how to select the appropriate model!!

6- 26

Your library will likely have several WLMs to be used for different design sizes. As the size of a design increases, standard cells can be physically placed further apart within that design, which means that, on average, wire lengths increase. It is therefore important to select the appropriate WLM for your design, based on size, to accurately model the interconnect RCs. Some library vendors may use obvious names for their WLMs, making it clear to the user which WLM to use, e.g. "*300kGates*, *600kGates*, etc."

In the examples above it looks like the model names *140000* and *8000000* correlate to some sort of size, but it is not clear what the size unit is. It could be the same unit as the area unit of this library or it could be a different unit, for example: the library area units may be may be *mils*^{2} while the WLM model names (140000 e.g.) relate to the number of transistors in the design. The vendor may choose to name the models "*WLM1*, *WLM2*, *WLM3*, etc", in which case it is really not clear which model to use for a particular design size. You should therefore find out from the library provider which WLM model should be used with which design size.

The units of resistance and capacitance are defined in the library: `report_lib <lib_name>`

If DC encounters a fanout count greater than the largest fanout listed in the model, it will use the extrapolation slope number to calculate the length. E.g., in the 140000 model above the length for a fanout of 22 is: 952.16 + 93.7215 * 2 = 1139.6 length units. The capacitance for this net is: 1139.6 * 0.000086 = 0.098 pF. The resistance is: 1139.6 * .000331 = 0.377 kΩ.

Specifying Wire Loads in Design Compiler

- **Manual selection:** `set_wire_load_model -name 8000000`
- ***Automatic model selection* selects an appropriate wireload model during compile:**

  (enabled by default if selection table available in library)

```
dc_shell-xg-t> report_lib 90nm_com
    Selection                      Wire load name
    min area    max area
   -------------------------------------------
        0.00     43478.00          140000
    43478.00     86956.00          280000
    86956.00    173913.00          560000
   173913.00    347826.00         1000000
     ...
```

- **To override automatic selection and select manually:**

```
set auto_wire_load_selection false
set_wire_load_model -name 8000000
```

6- 27

By default the `set_wire_load_model` is applied to all nets in the current design as well as in and between sub-designs.

Your technology library may have a default wire load model specified, which will be used if no wire load model is manually or automatically applied. To find out what the default model is use one of the following commands:

dc_shell-xg-t> get_attribute *<lib_name>* **default_wire_load**

dc_shell-xg-t> report_lib *<lib_name>*

You may want to override automatic model selection and manually specify the model. Here are a couple of advantages of doing so:

1) Your design is such that a more conservative WLM is required than what the automatic wireload selection table would choose (e.g. your design is actually a piece of a larger design so the area of your design is not really indicative of the overall design area)
2) If compile time is an issue turning off automatic model selection and manually selecting it may improve run time somewhat

Wire Delay Calculations and Topology

WLM determines one R and one C value for each net – no delays

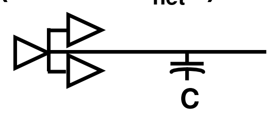
The *tree-type* determines R and C distribution for timing calculations. Interconnect delay (D_C) is measured from state change at driver pin to state change at each receiving cell's input pin (same in every branch!).

`best_case_tree`

Optimistic

Load adjacent to driver: $R_{net} = 0$ (but not C_{net}!!)

$D_C = 0$



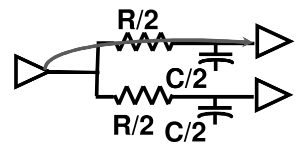
`balanced_tree`

Default

Each load pin shares equal portion of net's R and C

$D_C = (R_{net} / N) \times (C_{net} / N + C_{pin})$

where N= fanout

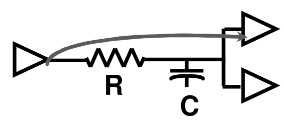


`worst_case_tree`

Pessimistic

Assume a lumped RC load

$D_C = R_{net} \times (C_{net} + C_{pins})$



6- 28

The **best-case** tree assumes the load is located next to the driver, so the net resistance, and therefore the net delays will be zero (even if the resistance per unit length parameter in the library is not zero). However, the net capacitance is not zero. This allows accurate cell delay and and cell transition calculations, independent of the zero net delay.

The **balanced-tree** models the case where all load pins are on separate but equal-length branches of the interconnect wire. Each load incurs an equal percentage of the total wire capacitance and wire resistance.

The **worst-case-tree** models the case where the load pins are at the extreme end of the wire. Each load incurs the full wire capacitance and wire resistance.

The `report_lib` command lists the operating conditions with their *Interconnect Models* defined in a technology library.

Tree-type Set by Operating Conditions

- **STA scales each cell and net delay based on process, voltage, and temperature (PVT) variations**
- **The tree-type or interconnect model is determined by the library supplier and is associated with each operating condition model**

```
Operating Conditions:

Name    Library    Process   Temp  Voltage  Interconnect Model
-------------------------------------------------------------------
BCCOM  90nm_com      0.90     0.00   1.98     best_case_tree
WCCOM  90nm_com      1.10    70.00   1.62     worst_case_tree
```

6- 29

"k-factors" may be defined in the library source code (by the vendor) to scale both cell parameters AND wire resistance/capacitance values for variations in process, voltage, and temperature. Many vendors do not change their wire delay scaling as PVT changes.

Wire Load Models vs 'Topographical Mode'

- **Wire load models (WLMs) are based on statistical averages and are not specific to *your* design**
- **In Ultra Deep Sub-Micron (UDSM) designs the interconnect parasitics have a *major* impact on path delays → need accurate RC estimates**
- **For UDSM designs WLMs are not adequate: It is recommended to use DC's 'Topographical Mode' (discussed in a later Unit)**
  - Topographical mode eliminates the use of WLMs
  - Uses placement algorithms under-the-hood to estimate wire lengths and parasitics
  - Provides much better timing correlation to that of the actual physical layout

6- 30

Summary: Commands Covered (1 of 2)

TOP.con

```
# Area and Timing constraints go here
...
...
set_max_capacitance $MAX_INPUT_LOAD $all_in_ex_clk
set_load [expr $MAX_INPUT_LOAD * 3] [all_outputs]
set_load 0.080  [get_ports OUT23]
set_load [load_of slow_proc/NAND2_3/A] [get_ports OUT42]
set_input_transition 0.12 [remove_from_collection \
                         [all_inputs][get_ports B]]
set_driving_cell -lib_cell FD1 -pin Q [get_ports B]
set_operating_conditions -max WCCOM
set_wire_load_model -name 1.6MGates
```

6- 31

Summary: Commands Covered (2 of 2)

MAX_CAP.tcl

```
# Algorithm to find largest max_capacitance in library and
# apply that value as a conservative output load

set LIB_NAME ssc_core_slow
set MAX_CAP 0
set OUTPUT_PINS [get_lib_pins $LIB_NAME/*/* -filter
"direction == 2"]
foreach_in_collection pin $OUTPUT_PINS {
   set NEW_CAP [get_attribute $pin max_capacitance]
   if {$NEW_CAP > $MAX_CAP} {
      set MAX_CAP $NEW_CAP
   }
}
set_load $MAX_CAP [all_outputs]
```

6- 32

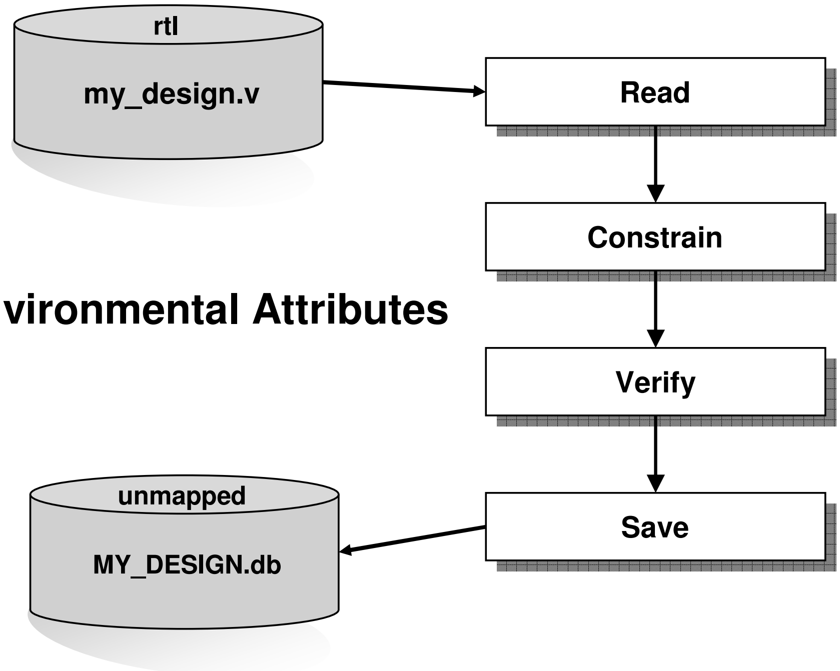
Summary: Unit Objectives

You should now be able to:

- **Apply environmental attributes to model the timing effects of:**
  - Input drivers and transition times
  - Capacitive output loads
  - PVT operating conditions or ‘corners’
  - Interconnect parasitic RCs

6- 33

Lab 6: Environmental Attributes



45 minutes

Apply Environmental Attributes

6- 34

Agenda

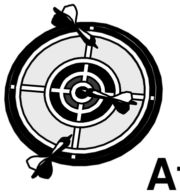
DAY 2

5	Partitioning for Synthesis	
6	Environmental Attributes	
7	Compile Commands	
8	Timing Analysis	
9	More Constraint Considerations	

Synopsys 10-I-011-SSG-013

7-1

Unit Objectives

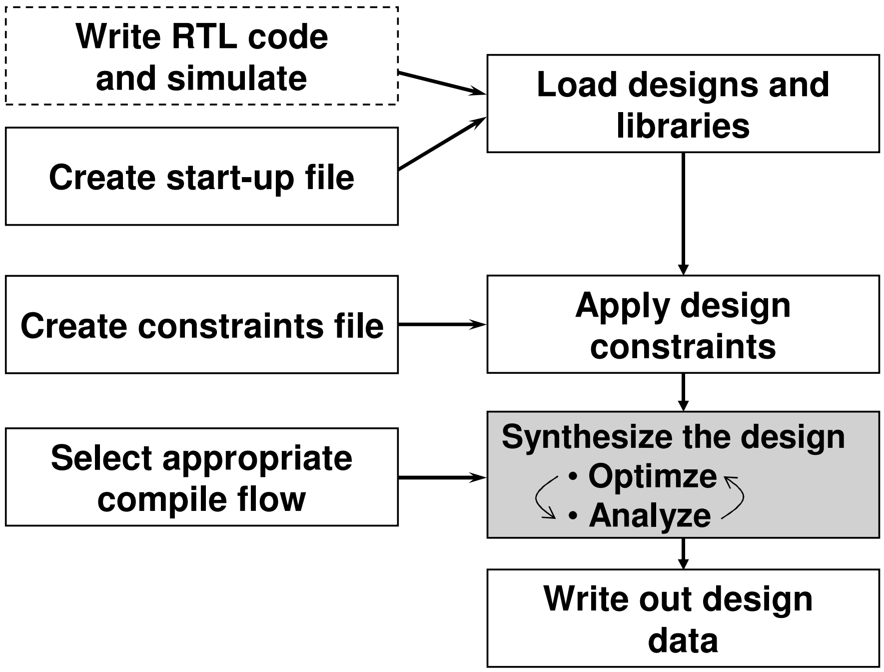


After completing this unit, you should be able to:

- **Select the recommended initial *compile command(s)*, based on license availability**
- **Describe what the recommended compile command *options* do**

7- 2

RTL Synthesis Flow



7-3

Compile Commands Covered in this Unit

run.tcl

```
# DC Expert  initial compiles
compile -boundary -scan -map_effort high
report_constraint -all_violators
compile -boundary -scan -map_effort high \
        -incremental  (-area_effort2 medium|low|none)

# DC Ultra  initial compile
compile_ultra -scan -retime -timing|-area
```

7- 4

Topographical Commands Covered

run.tcl

```
# Specify physical libraries for Topographical mode
create_mw_lib
         -technology            <technology_file> \
         -mw_reference_library <mw_reference_libraries> \
                                <mw_design_library_name>
open_mw_lib                     <mw_design_library_name>
set_tlu_plus_files \
                -max_tluplus    <max_tluplus_file> \
                -tech2itf_map   <mapping_file>
```

floorplan.con

```
# Physical constraints which define the floorplan for Topographical mode
set_aspect_ratio               set_port_side
set_utilization                set_port_location
set_placement_area             set_cell_location
set_rectilinear_outline        create_placement_keepout
```

7- 5

Requirements for Good Synthesis Results

- **The following are *crucial* in obtaining good synthesis results**
  - Good quality RTL code - optimum for DC
  - Complete and accurate constraints

See Appendix 1 for coding style examples

- **To greatly simplify the synthesis flow, and also achieve the best possible QoR**
  - Compile "top-down"

In this workshop the assumption is that you are starting with good quality RTL code and constraints, and are performing top-down compiles.

7- 6

Good quality RTL code - optimum for DC
- RTL follows DC's coding style guidelines and best practices
- RTL algorithms have been written in an efficient manner, with regards to the hardware that is inferred by the code

Complete and accurate constraints
- Realistic constraints and attributes, not over-constrained
- All false- or multi-cycle paths are identified
- Wire load models reflect physical floorplan and placement

Compile "top-down"
- Read, constrain and compile the top-level design, when possible

Compile Commands

Two commands are available to synthesize (optimize, compile) an RTL design to gates

- **`compile`**
  - Available with DC Expert as well as DC Ultra licenses
- **`compile_ultra`**
  - Available only with a DC Ultra plus a DesignWare license

The recommended *compile* options are discussed next

7- 7

Unit Agenda

DC Expert two-pass compile commands

- **`compile`** options

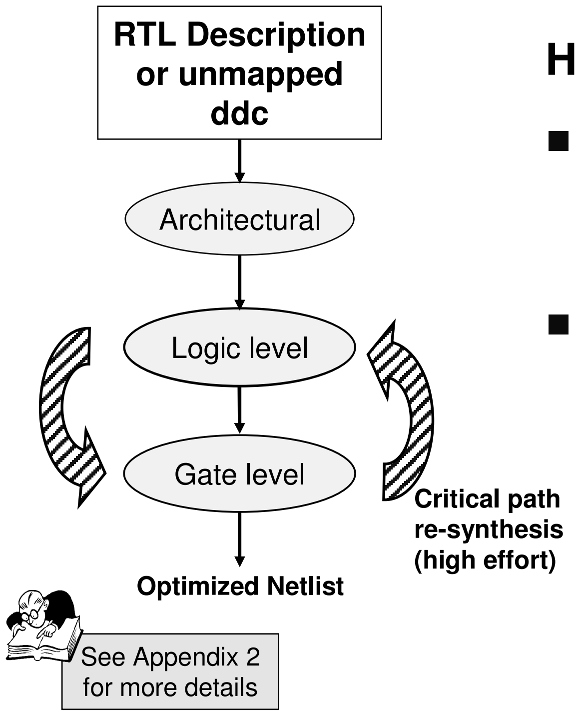
DC Ultra compile command

- **`compile_ultra`**
  - Ultra versus Expert
  - Topographical versus WLM mode

7- 8

DC Expert: First of Two-Pass Compiles

```
compile -boundary -scan -map_effort high
```



High map effort:

- **Applies maximum optimization effort during gate-level optimization**
- **Invokes 'Critical Path Resynthesis' as needed**
  - Cones of logic containing stubborn, violating critical paths are returned to logic-level optimization and then re-mapped, iteratively.

7-9

Map Effort Recommendation

Using *high map effort* allows DC Expert to invoke the most powerful available optimization algorithms, as needed.

Always enable *high map effort* compiles.

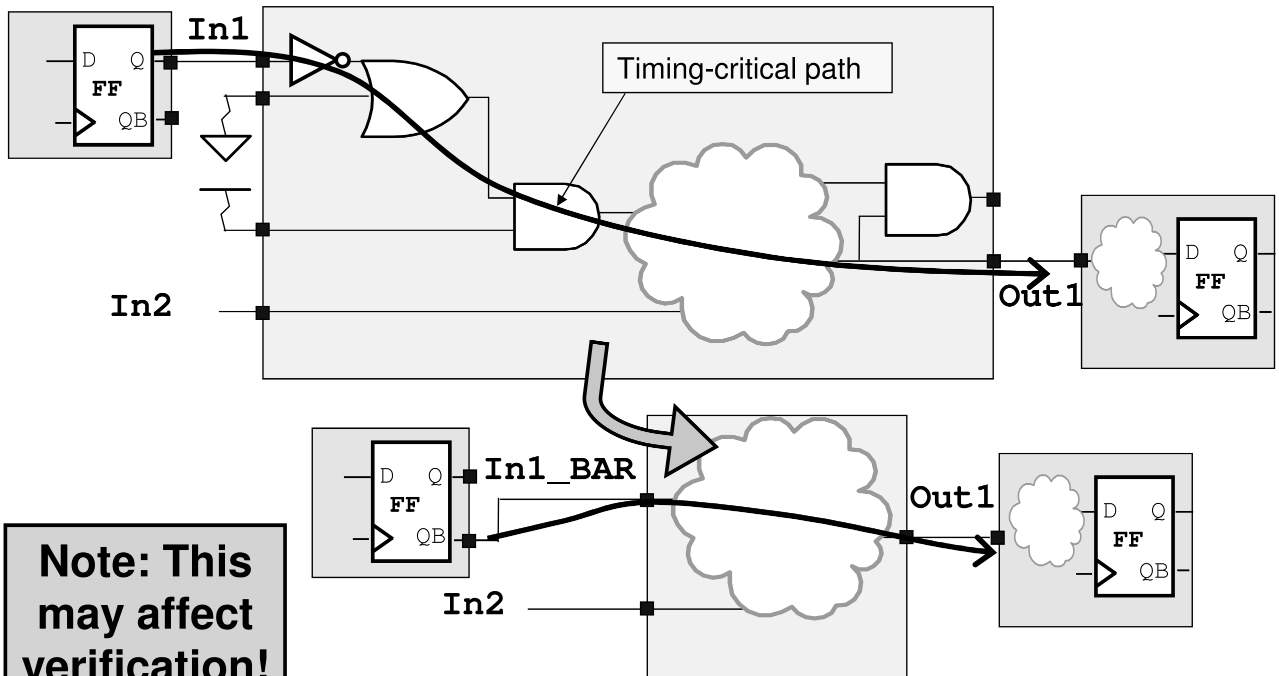
DC Ultra: Enabled automatically with `compile_ultra`

DC Expert: `compile –map_effort high`

7-10

Boundary Optimization

```
compile -boundary -scan -map_effort high
```



Note: This may affect verification!

7-11

Boundary optimization propagates constants, unconnected pins, and complement information during *compile* to reduce delay and area. DC does not perform boundary optimization by default.

Complemented pins are automatically renamed with an “_BAR”. This default behavior can be modified with `set port_complement_naming_style %s_SOMETHING_ELSE`.

Since boundary optimization may eliminate output ports and invert input port signals, this can have an impact on verification: For example, if a verification test-bench was created for the RTL code, which probes or drives one of these ports block ports which is subsequently inverted or removed, the existing test-bench will no longer work. If you know in advance which blocks must be preserved for verification purposes, use `set_boundary_optimization` to enable boundary optimization only on selective blocks.

Note: A file called *default.svf* is created automatically during compile, which records the "changes" that *boundary optimization*, *register repositioning* and *ungrouping* make to the design. This file is readable by Formality, Synopsys’ formal verification or equivalency checking tool. To rename the file use `set_svf <My_name.svf>`; invoke this command before reading in RTL (can put it in your *.synopsys_dc.setup* file). If using a third-party formal verification or equivalency checker try using `set_vsdc <My_name.vsdc>` to write out an ASCII "vsdc" file, which is intended to be readable by third party tools. Since this is a relatively new command and format the file may not be readable by all formal verification tools.

Boundary Optimization Recommendation

Boundary optimization can reduce delay and area. If you can handle the potential verification impact and the slightly longer run-time – it is worth it!

Always enable boundary optimization.

DC Ultra: Enabled automatically with `compile_ultra`

DC Expert:

```
compile -boundary; # affects entire design
```

or

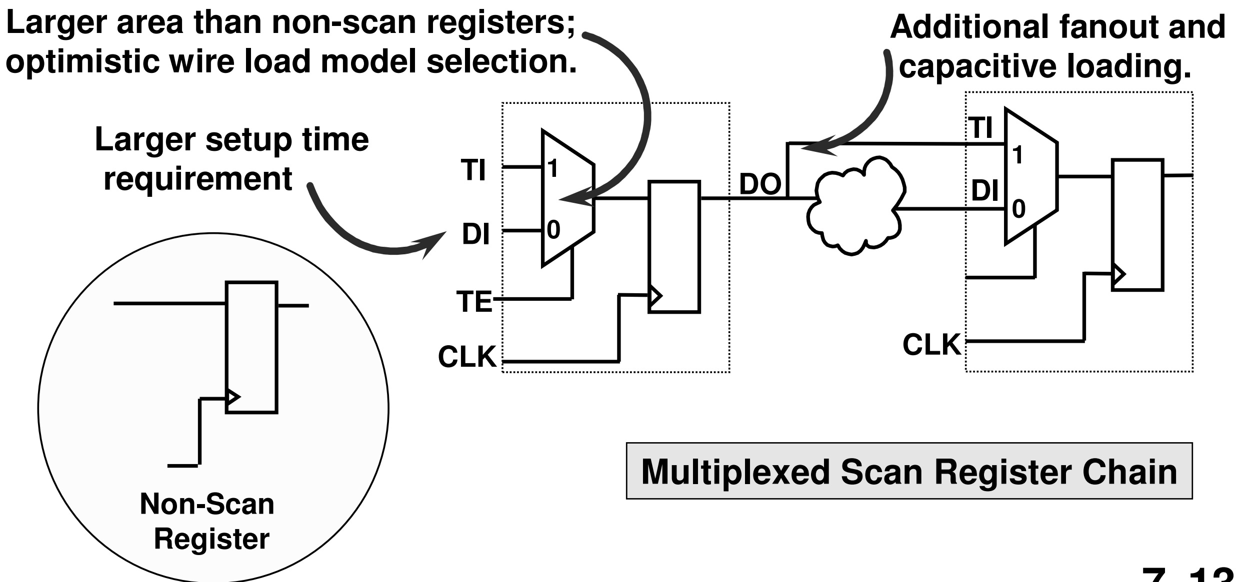
```
set_boundary_optimization <block_names> true
compile; affects selective sub-blocks
```

7-12

Scan Registers: The Problem

```
compile -boundary -scan -map_effort high
```

If you plan to insert scan chains, you must account for the impact of scan registers on a design *during* synthesis, in order to avoid negative surprises *after* scan insertion.

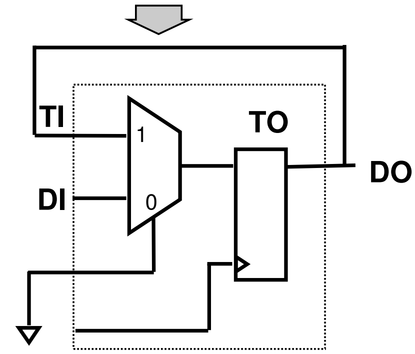


Multiplexed Scan Register Chain

7-13

Test-Ready Synthesis - The Solution

Scan register is used during initial compile, but not chained!



Benefits

Accurate area, timing, and loading modeled up front.

Easier synthesis flow -- scan cell insertion performed in one compilation step.

- **Include the scan style in the constraint script file:**
  - `set_scan_configuration -style \`
    `<multiplexed_flip_flop | clocked_scan`
    `| lssd | aux_clock_lssd>`

- **Test-ready synthesis requires a DFT Compiler license**

See Appendix 3 for more details

7- 14

The multiplexed_flip_flop style is the default – if this is your scan style you do not have to apply this command.

Test-ready Synthesis Recommendation

If you plan to insert scan in your design, and you have a DFT Compiler license:

Always perform *test-ready* compiles and incremental complies.

DC Ultra:

```
compile_ultra ... -scan
```

DC Expert:

```
compile ... -scan
compile ... -incremental -scan
```

7- 15

Generate a Constraint Report After Compile

```
compile -boundary -scan -map_effort high
report_constraint -all_violators
```

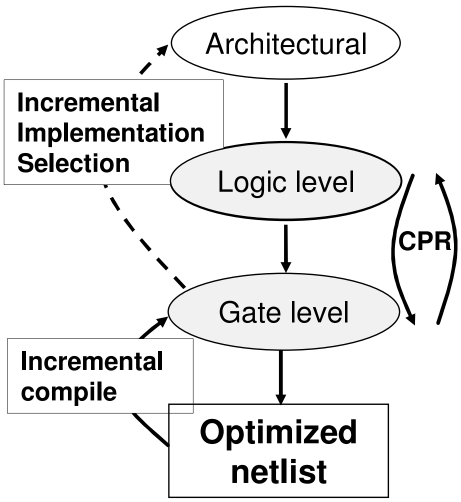
```
max_delay/setup ('clk' group)

                         Required        Actual
  Endpoint              Path Delay     Path Delay    Slack
  -----------------------------------------------------------------
  I_OUTBlk/z_reg[9]/D       4.19           5.43 r       -1.24  (VIOLATED)
  I_OUTBlk/z_reg[9]/D       4.19           5.32 f       -1.13  (VIOLATED)
  Out_7                     2.30           2.66 f       -0.36  (VIOLATED)

max_area
                         Required        Actual
  Design                   Area            Area      Slack
  -----------------------------------------------------------------
  TOTO                     2100          2190.53     -90.53  (VIOLATED)

max_capacitance
                       Required      Actual

Net                   Capacitance  Capacitance      Slack
---------------------------------------------------------------
CurrentState[0]            0.20        0.24          -0.04  (VIOLATED)
```

7-16

After any `compile` or *optimization* step, this is the recommended first report to run. If this report shows that there are no violations, no further timing and DRC analysis is needed. If there are violations, more detailed analysis can be performed with `report_timing`.

DC Expert: Second of Two-Pass Compiles

```
compile -boundary -scan -map_effort high \
  -incremental -area_effort medium|low|none
```



If you do not meet your constraints after the first compile:

- **Follow with an *incremental* high map-effort compile**
- **Can reduce run time by lowering area-effort**
  - By default area_effort = map_effort

7- 17

With an incremental compile DC begins re-optimizing the design at the Gate level. DC will perform Incremental Implementation Selection as needed to obtain faster arithmetic components. The high map-effort enables "Critical Path Resynthesis" (CPR) which takes a critical path, along with its surrounding cone of logic, and reverts the gates back to GTECH to re-optimize it (structuring) at the Logic level. CPR is invoked iteratively until all violating paths can no longer be improved, or meet timing.

DC Expert Recommendation

If you do not have a DC Ultra license:

Always start with the two-pass `compile` strategy

DC Ultra: Not applicable

DC Expert:

```
① compile -boundary -scan -map_effort high
  report_constraint -all_violators
② compile -boundary -scan -map_effort high \
    -incremental -area_effort medium|low|none
```

7- 18

Unit Agenda

DC Expert compile command

- **`compile`** options

DC Ultra compile command

- **`compile_ultra`**
  - Ultra versus Expert
  - Topographical versus WLM mode

7- 19

DC Ultra: One Command – Full DC Strength

```
compile_ultra -scan -retime -timing|-area
```

- **The full strength of Design Compiler in a single command**
- **Significantly better results possible for timing-critical high performance designs, as well as area-critical designs**
  - Invokes additional high performance optimization techniques not available with DC Expert (some examples follow)
  - Use `-retime -timing` for timing-critical designs
  - Use `-area` for non timing-critical designs
- **Requires an Ultra license as well as a DesignWare license**

If you have the licenses make this the first compile!

7- 20

The `compile_ultra` command automatically deploys a two-pass compile strategy.
With `-timing` (short for `timing_high_effort_script`) it invokes timing-centric, high-performance arithmetic algorithms, including (See some examples on the next few pages):

- Timing driven high-level optimization (HLO)
- Macro architecture optimization for arithmetic operations
- Selection of best datapath implementations from the DesignWare library
- Ungrouping of non-pipelined DesignWare parts
- Boundary optimization
- Wide-fanin gate mapping to reduce levels of logic
- Aggressive logic duplication for load isolation
- Auto-ungrouping of hierarchies along the critical paths
- DFT flow support (Test-ready compile using the "-scan" option)

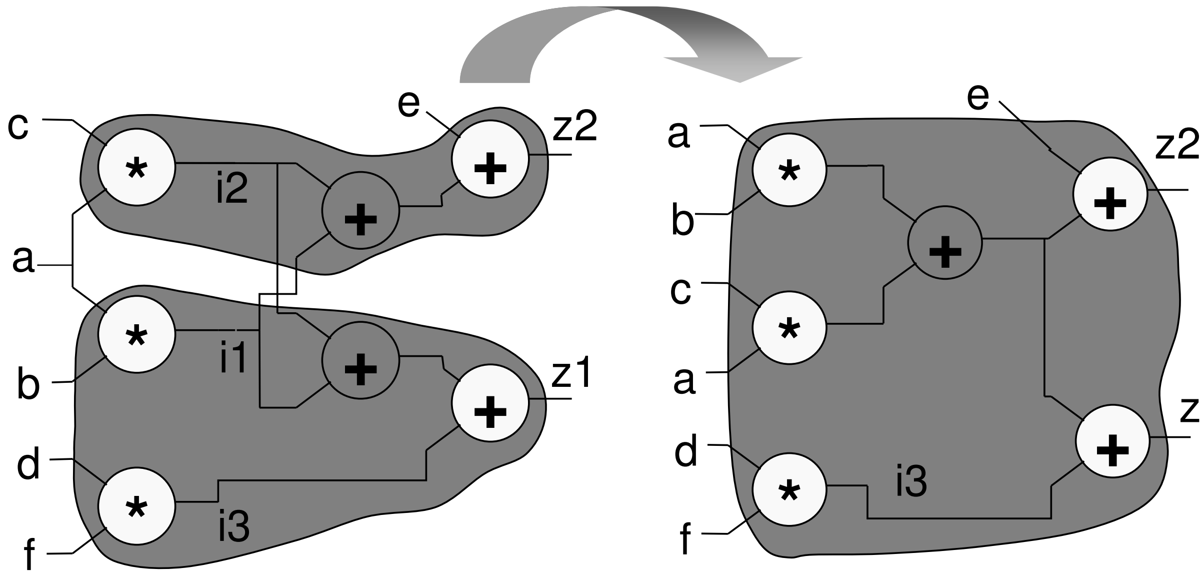
The `-retime` option is available starting with v2007.03. It performs "adaptive register retiming", which moves the logical location of registers up or down a timing path, to help improve local critical path timing without creating or worsening timing violations of surrounding paths. This will be discussed further in a later unit, in conjunction with the `optimize_registers` command.

The `-area` option (short for `area_high_effort_script`) invokes optimization algorithms which are optimum for area reduction – use this option for non timing-critical designs.

The `-timing/-area` options invoke a 'script' under the hood which executes the best-known, proven synthesis techniques and algorithms for a particular DC release. As additional and/or better techniques and algorithms are developed in future releases, the underlying scripts will be updated to reflect the latest and greatest ones, and the user will not need to change the command or the option.

Note: If you have an older script which includes some of the following pre-`compile_ultra` commands and variables, they are obsolete and should be removed from your script: `partition_dp`; `transform_csa`; `set dw_prefer_mc_inside true`; `set compile_new_optimization true`.

Ultra Optimization: Operator Merging

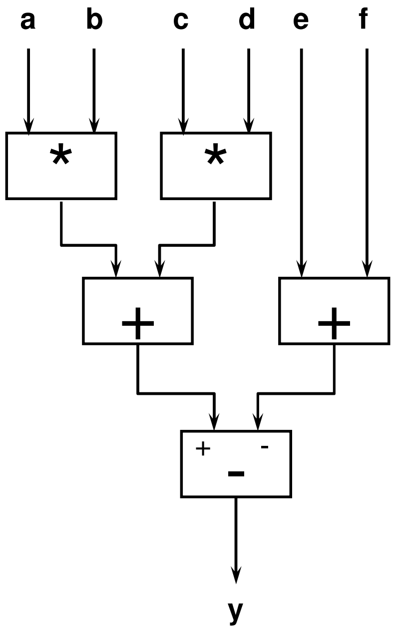


Arithmetic functions with shared sub-expressions are grouped into a single tree – smaller circuit

7- 21

Ultra Optimization: CSA Transformations

y <= a * b + c* d - e - f

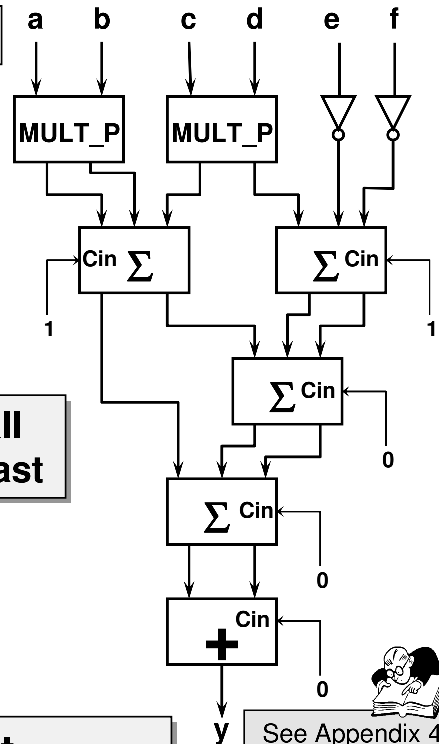


Small and slow

or

Fast and large

Small and fast



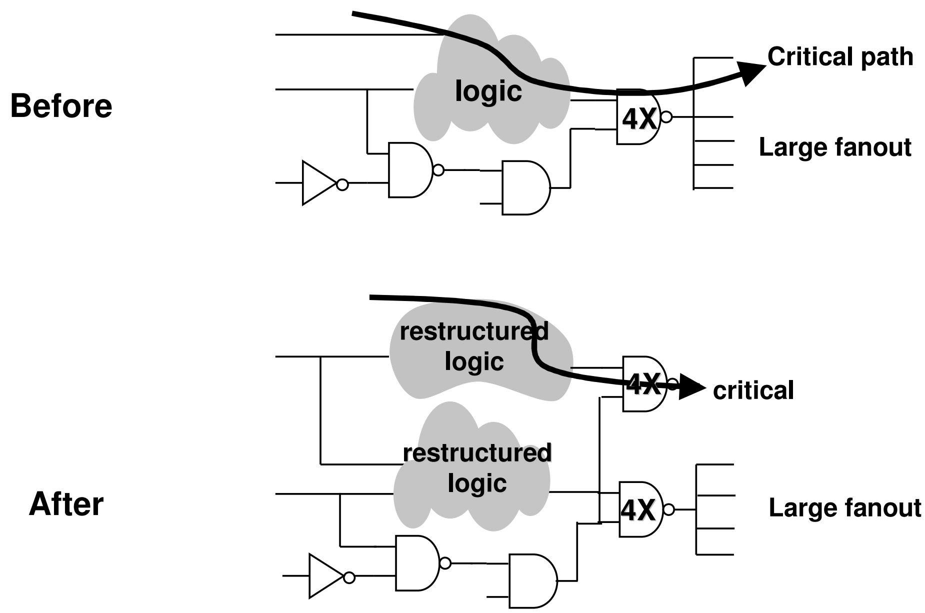
CSA transformations produce the most efficient arithmetic circuit – small *and* fast

See Appendix 4 for a detailed explanation

7- 22

Multipliers, adders and subtractors are transformed using Carry Save Adder (∑) trees: CSA adders are as small as ripple adders, but have no carry-ripple delay – they are small and fast.

Ultra Optimization: Logic Duplication

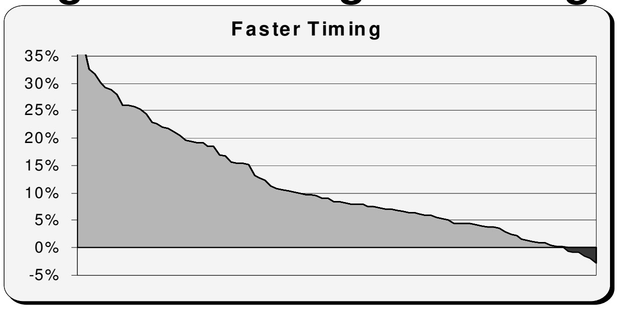


Load-splitting and logic duplication reduces critical path delays – faster (but larger) circuit

7- 23

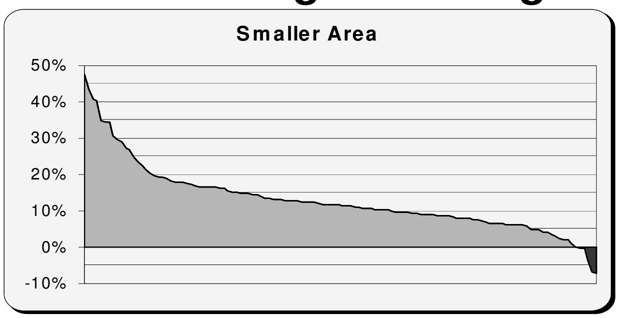
compile_ultra versus *2-pass* compile

- **Timing-critical designs: Average 9% faster**



140 customer designs

- **Area-critical designs: Average 14% smaller**



200 customer designs

Results are even better using `-retime -timing` or `-area`

- **No runtime overhead**

7- 24

Comparisons based on compile_ultra vs. a “two-pass high-effort expert compile”, V2007.03.

(Note that the above compile_ultra runs were performed without the –timing or –area high effort scripts, and without `-retime`. With these options even better results are seen.)

Designs include a 64-bit DSP, RISC core, network processor, storage filter, configurable router.
Design Sizes: 200K to 2M synthesizable gates
Clock Speeds: 100 MHz to 500 MHz.

There are no known “design characteristics” which can help predict how much QoR improvement a particular design is likely to achieve using Ultra.

Topographical Mode

```
UNIX% dc_shell-t -topographical
...
dc_shell-topo> compile_ultra -scan -retime -timing
```

- **The `-topographical` option invokes DC in *topographical mode* (versus *WLM* or *wireload mode* without the option)**
- **`compile_ultra` in *Topographical Mode* (vs *WLM Mode*) reduces iterations and time-to-results**
  - Performs *placement* under-the-hood to estimate wire lengths – no WLMs
  - Provides better correlation with the placed design's timing – no surprises
  - Provides a better starting netlist for Placement – fewer iterations
- **`compile_ultra -incremental` available for second compile[1]**
- **Requires physical libraries (Milkyway), in addition to the logical libraries**

7- 25

Topographical mode is only available in conjunction with the `compile_ultra` command, and hence requires the same licenses. The `compile` command can not be invoked in topographical mode.

Starting with DC v2007.03 `compile_ultra -incr` can be invoked in both WLM and Topographical mode.
In DC v2006.06 the `compile_ultra -incr` command can only be invoked in topographical mode. In WLM mode, if an incremental compile is required after the initial `compile_ultra`, you must use `compile -incr -map high ... .`

Specifying the *Milkyway* Libraries

First DC session

run.tcl

```
create_mw_lib
        -technology           <technology_file> \
        -mw_reference_library <mw_reference_libraries> \
                              <mw_design_library_name>
open_mw_lib                   <mw_design_library_name>
set_tlu_plus_files \
              -max_tluplus   <max_tluplus_file> \
              -tech2itf_map  <mapping_file>
```

Subsequent DC sessions

run.tcl

```
open_mw_lib                   <mw_design_library_name>

set_tlu_plus_files \
              -max_tluplus   <max_tluplus_file> \
              -tech2itf_map  <mapping_file>
```

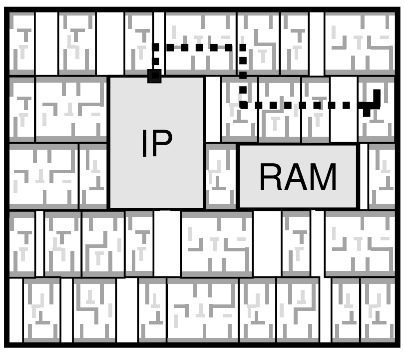
See Appendix 5 for more background info

7- 26

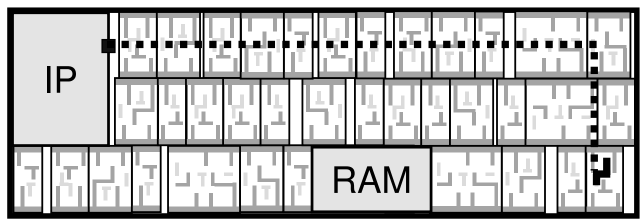
The physical Milkyway reference libraries (e.g. standard cell, IP/Macro cell and/or pad cell libraries) contain the physical layout description of the cells in the synthesized netlist, which are used when the topographical compile performs under-the-hood placement. The technology file defines the process metal layers, physical design rules, units of resistance, capacitance, etc. The TLU-plus files define models for calculating ultra-deep-submicron RC parasitic values from extracted wire data. These files and libraries are provided by the vendor or library group.

Default versus Actual Floorplan

- **Topographical mode uses a *default* floorplan, if none is specified**
  - Can achieve even better correlation if the design's actual floorplan is defined



DC-Topo placement and interconnect estimate using a default floorplan



Actual floorplan and post-DC placement

7- 27

If no floorplan information is provided, the topographical compile:
- Assumes a square placement area (aspect ratio = 1)
- Assumes a 60% utilization (leaving 40% of "empty" space in the core)
- Places the IP/macro cells along with the standard cells
- Assigns arbitrary pin locations to the IP/macro cells

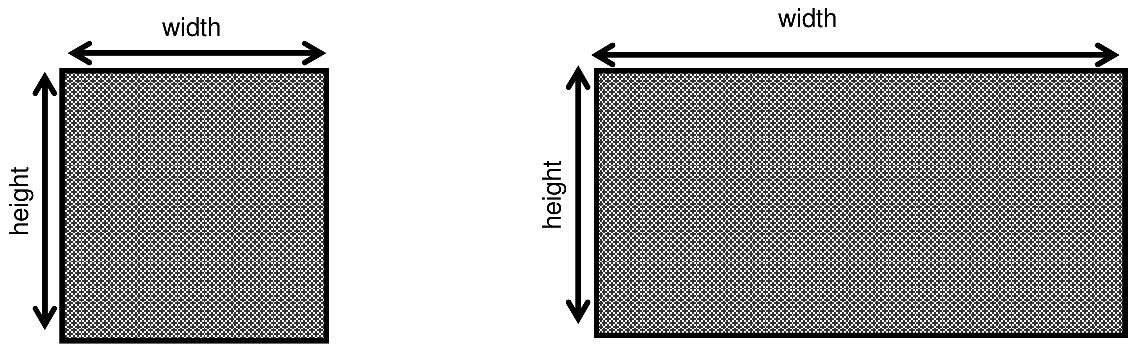
$$\text{Utilization} = \frac{\text{(Total Std Cell + Macro Cell Area)}}{\text{Core Placement Area}} \times 100\%$$

In most cases a floorplan of the chip is created prior to placement, during which the shape (aspect ration) and size (utilization), as well as IP/macro cell placement and pin locations are all defined.

In the example above, since the the actual floorplan is very different from the default floorplan assumed by DC Topographical, DC's interconnect RC and delay estimate of one particular net does not correlate well with the actual net's RC and delay.

Defining Relative Core Shape: Aspect Ratio

- **Aspect ratio is the height to width ratio of a block**
  - Defines the block shape
  - Default aspect ratio is 1



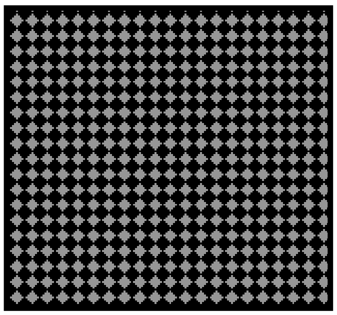
`set_aspect_ratio 1`

`set_aspect_ratio 0.5`

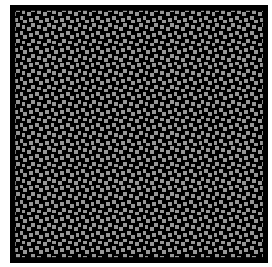
7- 28

Defining Relative Core Size: Utilization

- **Utilization dictates how densely you want your cells to be placed within the block**
  - Increasing utilization will reduce the core area
  - Default Utilization is 0.6



```
set_utilization 0.6
```



```
set_utilization 0.85
```

7- 29

All the Supported Physical Constraints

Core Area

Relative Constraints

`set_aspect_ratio` **and**

`set_utilization`

Exact Constraint

`set_placement_area` **or**

`set_rectilinear_outline`

Ports

Relative Constraint

`set_port_side`

Exact Constraint

`set_port_location`

Macros

Exact Constraint

`set_cell_location`

Placement Blockages

Exact Constraint

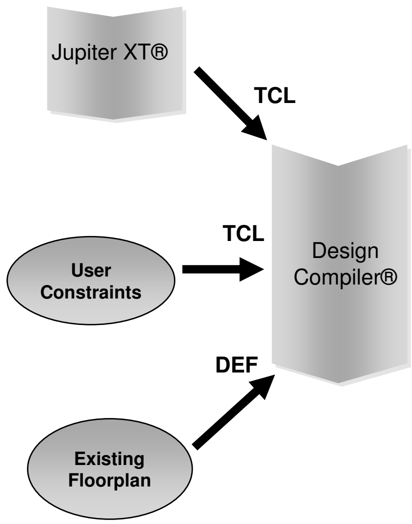
`create_placement_keepout`

See Appendix 6 for more examples

7- 30

Exact constraints have higher precedence over relative constraints, if both are applied.

Applying Physical Constraints



Physical constraints can be:

- **Generated directly from Jupiter XT** [1]
  `source <generated_TCL_file>`
- **Provided as manual input by the user**
  `source <user_created_TCL_file>`
- **Automatically extracted and applied from an existing floorplan (DEF)** [2]
  ```
  extract_physical_constraints \
                    <DEF_file>
  ```
- **Explicitly saved after being changed by auto-ungrouping or** `change_names` [3]
  ```
  write_physical_constraints \
         -output PhysConstr.tcl
  ```

7- 31

Note: If no physical constraints are applied in DC Topographical mode, DC assumes a basic square core placement area with a default utilization percentage of 60%, and no macros or placement blockages. Therefore, if the floorplan is available it is highly recommended to apply the physical constraints.

[1] The physical constraints file is generated directly from Jupiter XT with `derive_physical_constraints`

[2] Extracted physical constraints are automatically applied to the current_design – no need to source anything. Extracted constraints consist of "exact" constraints only – no "relative" constraints.

[3] After `compile_ultra` your physical constraints may change due to auto-ungrouping or `change_names`. Starting with DC v2007.03 the physical constraints are stored in *ddc*, so the updated information is automatically passed on to IC Compiler. However, the physical constraints are not stored with the *ddc* file prior to DC v2007.03, so if you exit DC in between compiles the physical constraints will need to be re-applied after reading in the *ddc* file. You should therefore write out the updated physical constraints with `write_physical_constraints \ -output <filename.tcl>` before exiting DC if using v2006.06. Example:

```
dc_shell-topo> read_verilog RTL.v
               extract_physical_constraints <def_file>
               compile_ultra -scan -retime -timing
               change_names -hier -rules verilog
               write_physical_constraints -output PhysConstr.tcl
               write -hier -format ddc -output netlist.ddc
               exit

UNIX% dc_shell-t -topographical
dc_shell-topo> read_ddc netlist.ddc
               source PhysConstr.tcl
               compile_ultra -incremental -scan
```

DC Ultra Recommendations

DC Ultra's powerful optimization algorithms can result in far better QoR compared to DC Expert.

If you have an Ultra license (and DesignWare) use
`compile_ultra -scan -retime -timing|-area` !

If you have the physical libraries specify them and invoke Topographical mode!

```
UNIX% dc_shell-t -topographical
```

In topographical mode: If you have a floorplan of your design apply the "physical constraints" or extract them from DEF

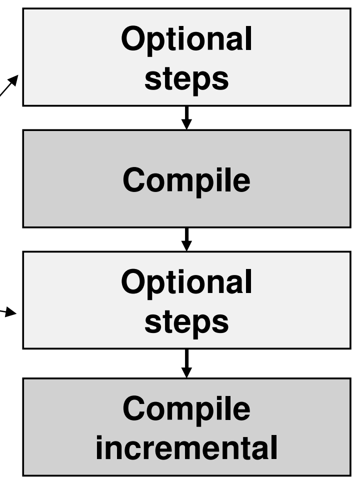
7-32

Using `compile_ultra`, in topographical mode, with a floorplanned design, is currently the best methodology for Synthesis, offering the highest chance of achieving the best possible QoR with the fewest number of iterations in the back end (physical Placement and Routing).

If you have an Ultra license but no DesignWare license (not recommended!), you will not be able to invoke `compile_ultra`. You can however enable a sub-set of the Ultra optimizations within the Expert flow (`compile` command) by setting certain attributes and variables. This "pseudo-Ultra" flow will not be discussed but is provided in the appendix of a later Unit for your reference.

Additional Synthesis Techniques

- **Besides the *compile* commands there are additional techniques to improve results in designs with:**
  - Arithmetic components
  - Pipelines
  - Poor hierarchical partitioning
  - Aggressive DRC requirements
  - Specific paths requiring more optimization focus



- **There is also a *parallel synthesis* technique if you need fast or frequent compile times for large designs**

7- 33

Compile Commands in Different Flows

DC Expert	DC Ultra
Additional steps	Additional steps
`compile …` + `compile -incr …`	`compile_ultra …`
Additional steps	Additional steps
`compile -incr …`	`compile_ultra -incr …`

The *additional steps* and the details of the different flows will be discussed in a later Unit

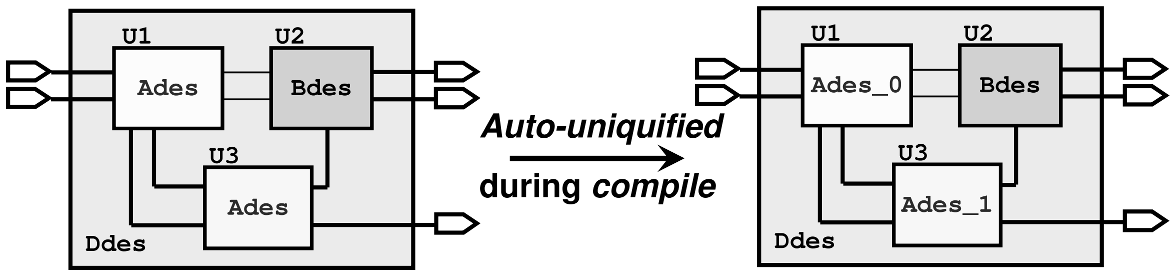
7- 34

Note: The "Additional steps" imply that you can include additional commands and variable settings at these locations in the flows. Each "Additional steps" box can have a different set of commands and variables, which will be shown in a later Unit.

Auto-uniquify : DC Version 2004.06 and Later

Since DC version 2004.06 the current design is automatically *uniquified* during compile

- Auto-uniquify happens during a later optimization phase, resulting in faster compiles compared to a manual `uniquify` *before* compile



Recommendation for DC version 2004.06 and later:

- Do not `uniquify` prior to compile
- If using an older script check for and remove `uniquify`

7-35

Note that you can still manually force the tool to uniquify designs before compile by executing the `uniquify` command, but this step contributes to longer runtimes.

You cannot turn off the auto-uniquify process.

Modify the variable `uniquify_naming_style` to control the naming convention for each copy of the multiply instantiated sub-designs.

Summary: *Compile* Commands Covered

run.tcl

```
# DC Expert  initial compiles
compile -boundary -scan -map_effort high
report_constraint -all_violators
compile -boundary -scan -map_effort high \
        -incremental (-area_effort2 medium|low|none)

# DC Ultra  initial compile
compile_ultra -scan -retime -timing|-area
```

7- 36

Summary: *Topographical* Commands Covered

run.tcl

```
# Specify physical libraries for Topographical mode
create_mw_lib
         -technology             <technology_file> \
         -mw_reference_library <mw_reference_libraries> \
                                 <mw_design_library_name>
open_mw_lib                      <mw_design_library_name>
set_tlu_plus_files \
                -max_tluplus     <max_tluplus_file> \
                -tech2itf_map    <mapping_file>
```

floorplan.con

```
# Physical constraints which define the floorplan for Topographical mode
set_aspect_ratio              set_port_side
set_utilization               set_port_location
set_placement_area            set_cell_location
set_rectilinear_outline       create_placement_keepout
```

7- 37

Summary: Unit Objectives

You should now be able to:

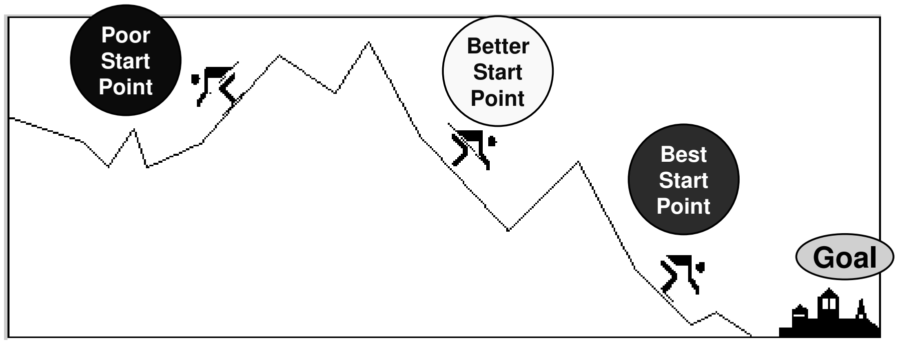
- **Select the recommended initial *compile command(s)*, based on license availability**
- **Describe what the recommended compile command *options* do**

7- 38

Appendix 1

RTL Coding Style Examples

The Importance of Quality Source Code

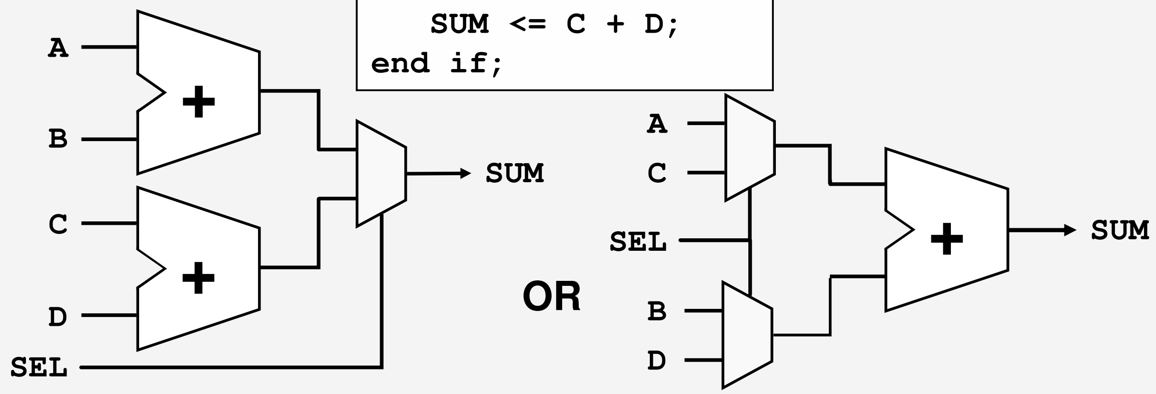


- **Functionally equivalent RTL code using different *coding styles* will give different synthesis results**
- **You cannot rely solely on Design Compiler to “fix” a poorly coded design!**
- **Understanding how DC interprets RTL coding style will enable you to get the best synthesis results → See following examples …**

7- 40

Example: Coding to Allow *Resource Sharing*

```
if (SEL = '1') then
   SUM <= A + B;
else
   SUM <= C + D;
end if;
```



Resource sharing not applied due to critical SEL→SUM delay

Resource sharing applied to save area

<u>DC coding style</u>: Arithmetic "resources" within an *if* or *case* statement will be considered for resource sharing. This allows DC to select the smallest architecture that meets timing

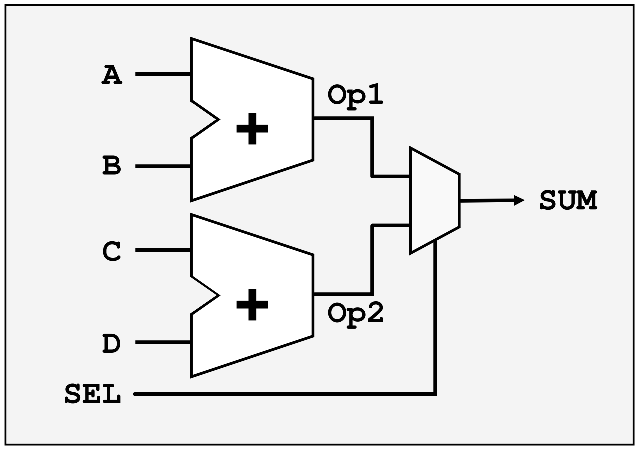
7- 41

Resource sharing decisions are made during the architectural level optimization phase of compile. The decision to share or not to share is based on two factors: 1) Coding style – only resources within the same *if* or *case* statement will be considered for sharing; 2) Constraints – sharing saves area but occurs only if it does not introduce or increase any path delay violations.

Actual circuit implementation depends on cells available in the target library and the constraints.

Example: Preventing Resource Sharing

```
Op1 = A + B;
Op2 = C + D;

if (SEL == 1'b1)
  SUM = Op1;
else
  SUM = Op2;
end if
```



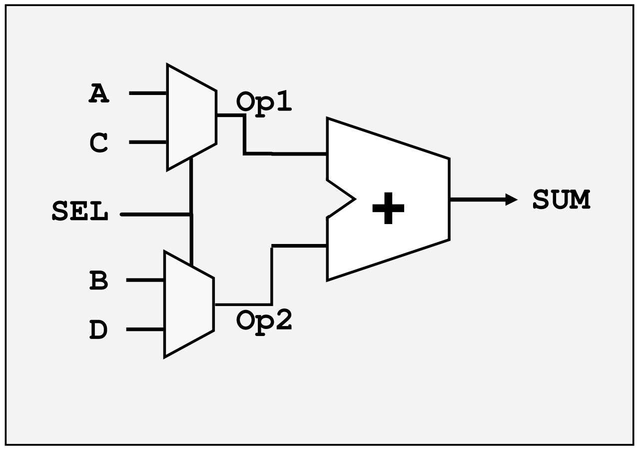
Since the arithmetic "resources" are outside the *if* statement no resource sharing occurs – the design may be much larger than necessary

7- 42

If you are not aware of the coding style which allows resource sharing to be considered by DC, you may code your design as above, which does NOT result in an area-efficient design.

Example: 'Forced' Resource Sharing

```
if (SEL == 1'b1)
  begin
    Op1 = A;
    Op2 = B;
  end
else
  begin
    Op1 = C;
    Op2 = D;
  end

SUM = Op1 + Op2;
```



Since the arithmetic "resource" is being shared outside the *if* statement DC can not "un-share" – Poor architecture if SEL→SUM is timing critical

Recommendation: Write code that lets DC decide if *resource-sharing* is warranted or not!

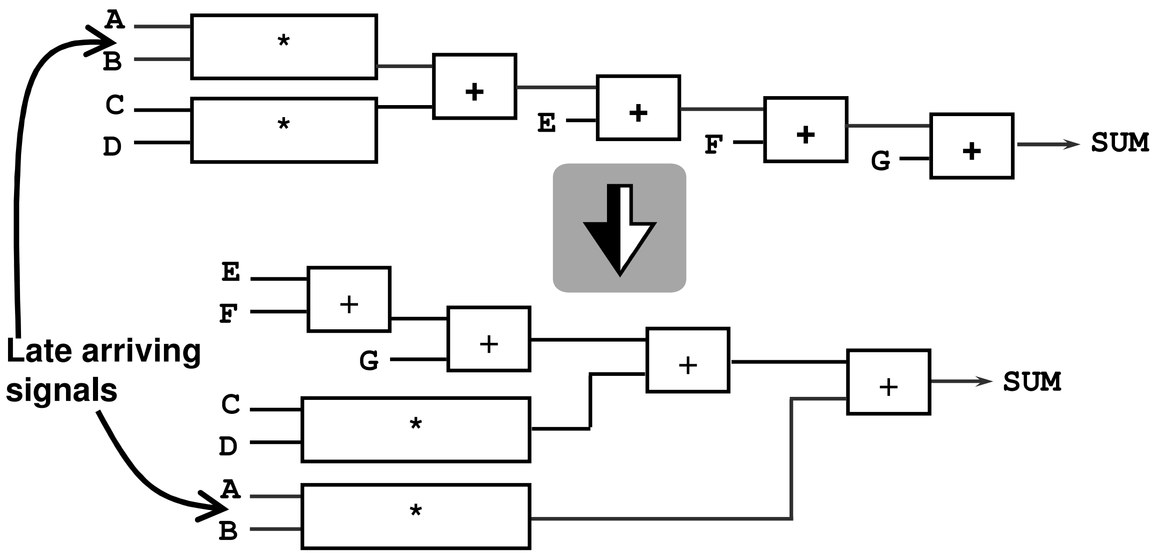
7- 43

By unknowingly coding as above you are in essence "forcing" a resource-shared architecture, which can not be un-done. If SEL→SUM is a timing-critical path, this design is NOT the best choice!

Conclusion:

Example: Coding to Allow Operator Reordering

```
SUM <= A*B + C*D + E + F + G
```

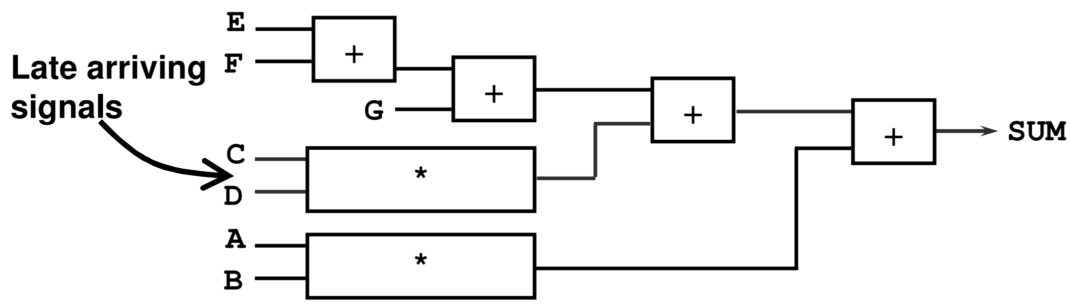


DC coding style: Operator reordering can re-arrange the order of arithmetic circuitry to meet timing requirements, as long as no ordering is 'forced' by using parentheses.

7- 44

Example: Preventing Operator Reordering

```
SUM <= (A*B) + ( (C*D) + ((E+F) + G) )
```



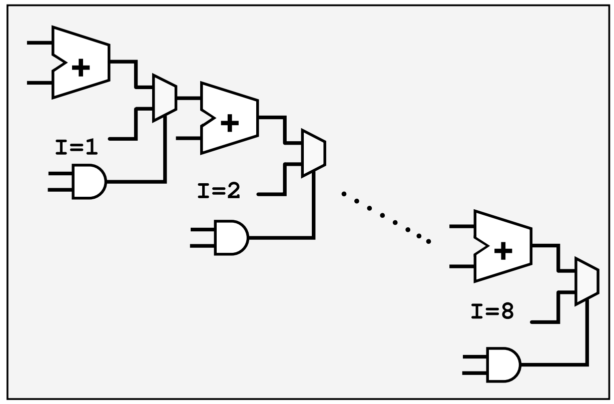
Because of the parentheses DC is not able to re-order the arithmetic operators –
Poor architecture if C/D→SUM are timing critical

Recommendation: Write code that <u>lets DC decide</u> if *operator-reordering* is warranted or not!

7- 45

Example: Poor ‘Algorithm’ using *For* Loop

```
/* Algorithm to find the LSB of IRQ
that is a 1, which determines what
offset is added to ADDR*/

/* Table lookup of offsets */
OFFSET[1] = 5'd1;  OFFSET[2] = 5'd2;
OFFSET[3] = 5'd4;  OFFSET[4] = 5'd8;
OFFSET[5] = 5'd16; OFFSET[6] = 5'd17;
OFFSET[7] = 5'd20; OFFSET[8] = 5'd24;

   DONE = 1'b0;
   ADDR = BASE_ADDR;
   for (I = 1; I <= 8; I = I + 1)
       if (IRQ[I] & ~DONE)
         begin
           ADDR = ADDR + OFFSET[I];
           DONE = 1'b1;
         end
```



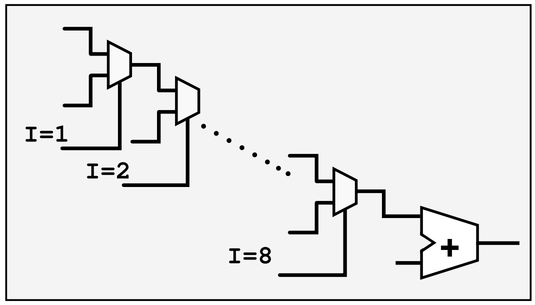
<u>DC coding style</u>: Logic in a *for* loop is ‘unrolled’ and duplicated `I` times. DC can NOT optimize away the duplicate logic – it can only make the duplicate logic as fast and small as possible.

7- 46

In the RTL ‘algorithm’ above the *OFFSET* addition is performed inside the *for-loop*, so the adder logic is repeated 8 times! Additionally, since the *for-loop* starts with the LSB of *IRQ* a *DONE* “flag” (additional logic) is required to determine when the LSB=1 condition is met, so that no further offset addition is performed. Note that since the *IRQ* input is not known in advance, the hardware must be built with all 8 adders to be able to handle any *IRQ* input value. DC can only make the adders as fast as possible, but the critical path will contain 8 adders.

Example: Better 'Algorithm' using *For* Loop

```
/* Determine the highest priority
interrupt line being asserted (the LSB
equal to 1), then "look up" the offset and
store as TEMP_OFFSET. Start at IRQ(8)
(*lowest* priority), and work UP to the
highest priority (LSB). TEMP_OFFSET of a
'lower' priority interrupt will be
overwritten by the offset for a higher
priority interrupt. ADD the offset AFTER
it is determined.
*/
OFFSET[1] = 5'd1;  OFFSET[2] = 5'd2;
OFFSET[3] = 5'd4;  OFFSET[4] = 5'd8;
OFFSET[5] = 5'd16; OFFSET[6] = 5'd17;
OFFSET[7] = 5'd20; OFFSET[8] = 5'd24;

 TEMP_OFFSET = 5'd0;

   for (I = 8; I >= 1; I = I - 1)
       if (IRQ[I])
           TEMP_OFFSET = OFFSET[I];

/* Calculate interrupt vector address */
   ADDR = BASE_ADDR + TEMP_OFFSET;
```



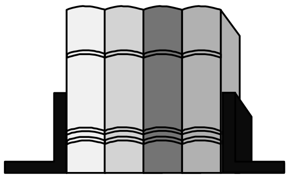
By modifying the "algorithm" used to determine the offset (exact same functionality) you end up with a much smaller and faster design!

Recommendation: Think about the hardware that your code implies – write efficient 'algorithms'

7- 47

By determining the *OFFSET* inside the *for-loop* but applying (adding) the *OFFSET* outside the *for-loop* you save the area and delay of 7 adders! By starting with the MSB instead of the LSB and taking advantage of the *priority* of *IF* statements, you can further eliminate the need for the *DONE* flag and its associated additional logic (AND-ing function), which was also repeated 8 times.

For More Information - Documentation



Synopsys On-Line Documentation on SolvNet

- HDL Compiler (Presto Verilog) Reference Manual
- HDL Compiler (Presto VHDL) Reference Manual
- Guide to HDL Coding Styles for Synthesis

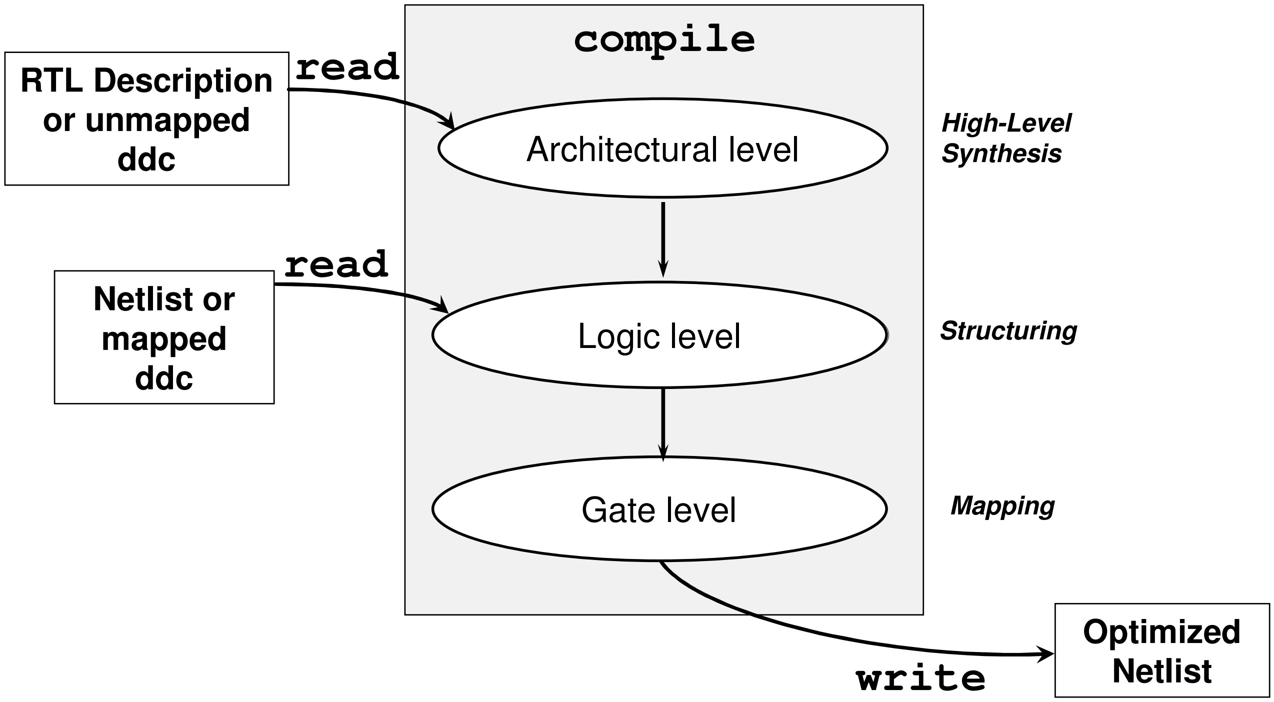
7- 48

Appendix 2

Three Levels of Optimization

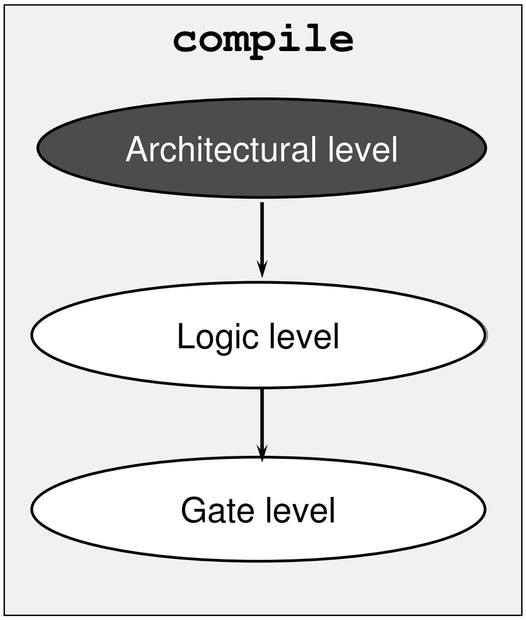
Compile: Three Levels of Optimization

Optimization can occur at each of three levels:



7-50

Architectural Level Optimization



High-Level Synthesis

- Resource Sharing
- Implementation Selection
- Reordering Operators
- Sharing Common Subexpressions

Some examples follow …

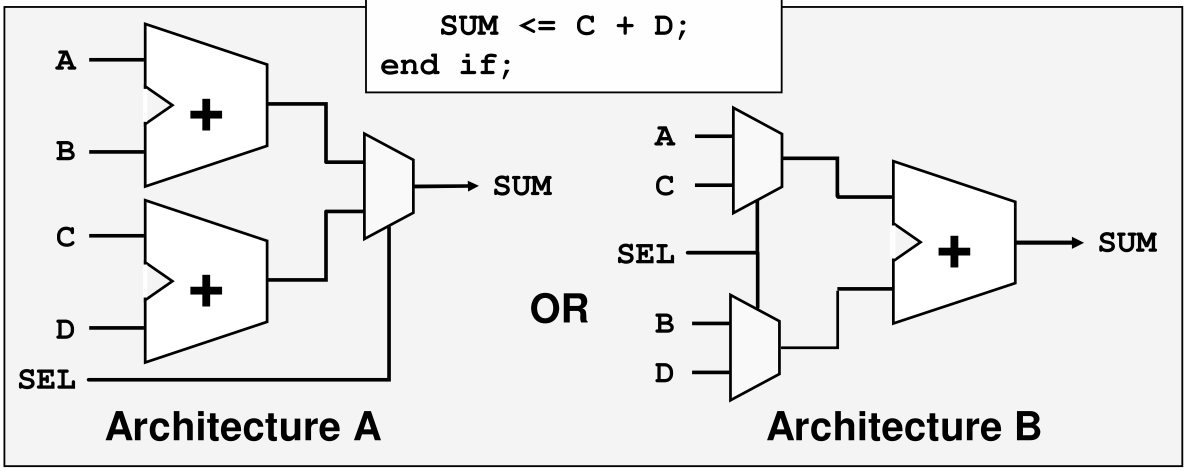
7- 51

These high-level synthesis decisions are made during compile. Before compile, while RTL is being read in (during the elaboration phase) some additional architectural manipulation takes place, including identifying sharable or merge-able operators.

Resource Sharing

This code can result in one of two architectures:

```
if (SEL = '1') then
   SUM <= A + B;
else
   SUM <= C + D;
end if;
```



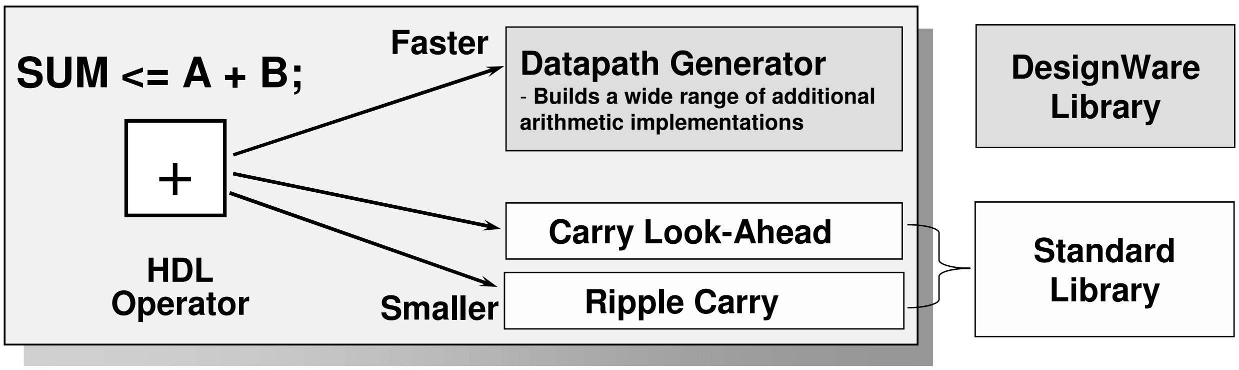
What type of adder should be synthesized?

7-52

The architecture decision above is made through the ‘high level synthesis’ or ‘architectural level optimization’ algorithm called “resource sharing”, based on coding style as well as timing constraints. If the SEL→SUM path is timing critical, the faster but larger Architecture A is selected, otherwise resources (DW parts) are shared to save area in Architecture B.

Implementation Selection

- **Multiple implementations for singleton arithmetic operators, available in the *Standard Library*, allow DC to evaluate speed/area tradeoffs and choose the best - the smallest that meets timing**



- **Once selected, each implementation is then optimized for the target technology library**
- **The 'High Performance' DesignWare Library allows more choices – discussed later**

7- 53

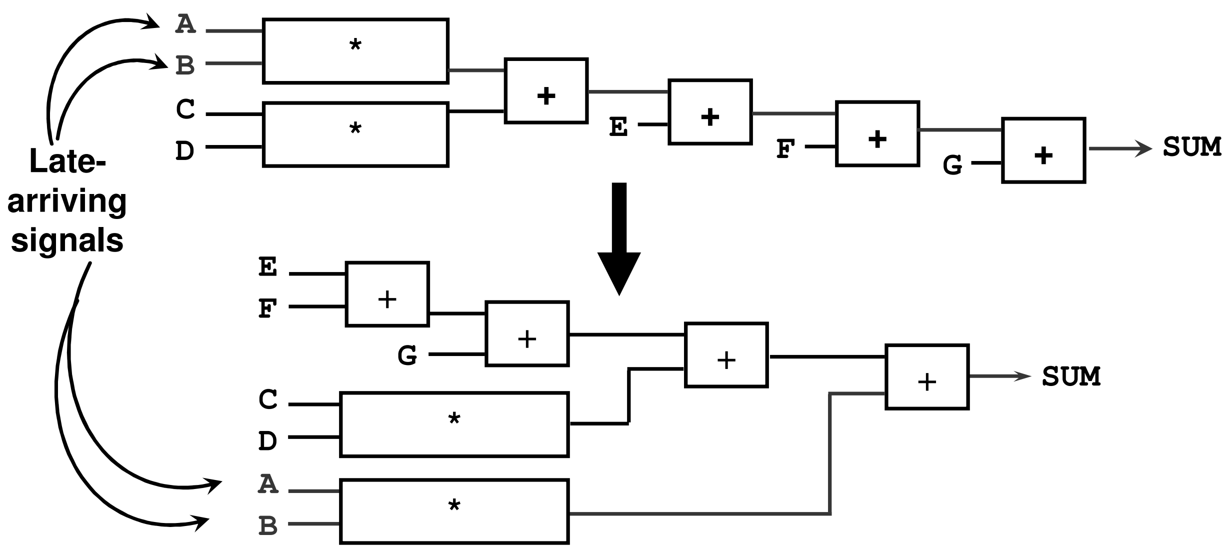
The *standard library* is automatically included, and set up for use with DC Expert.

There is also a DesignWare library available, which gives DC many more high-performance implementation possibilities arithmetic operators (including a datapath generator for adders, subtractors and multiplier), as well as an immense collection of standard *soft IP* blocks that can be instantiated in designs and optimized for the target technology. Access to the DesignWare library requires a DesignWare license, as well as a couple of variables settings. Discussed further in a later Unit.

Operator Reordering

This code implies the following initial operator ordering:

```
SUM <= A*B + C*D + E + F + G
```

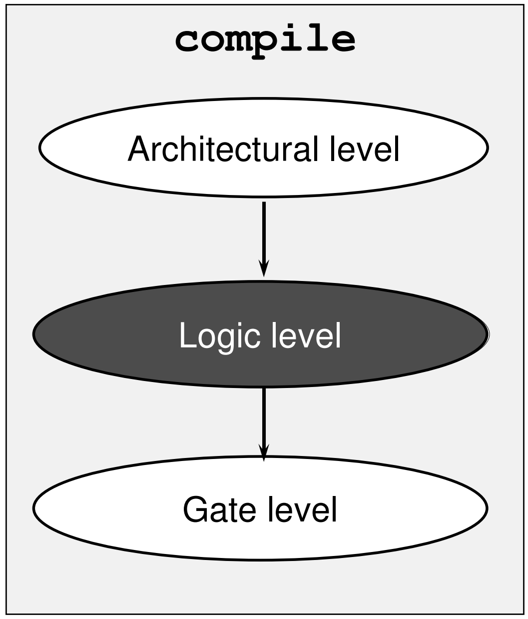


Operators are re-ordered to reduce critical path delays

7-54

All HLS or Architectural level optimization decisions are based on coding style as well as timing constraints. In general DC tried to build the smallest architecture that meets timing.

Logic Level Optimization



Structuring

7- 55

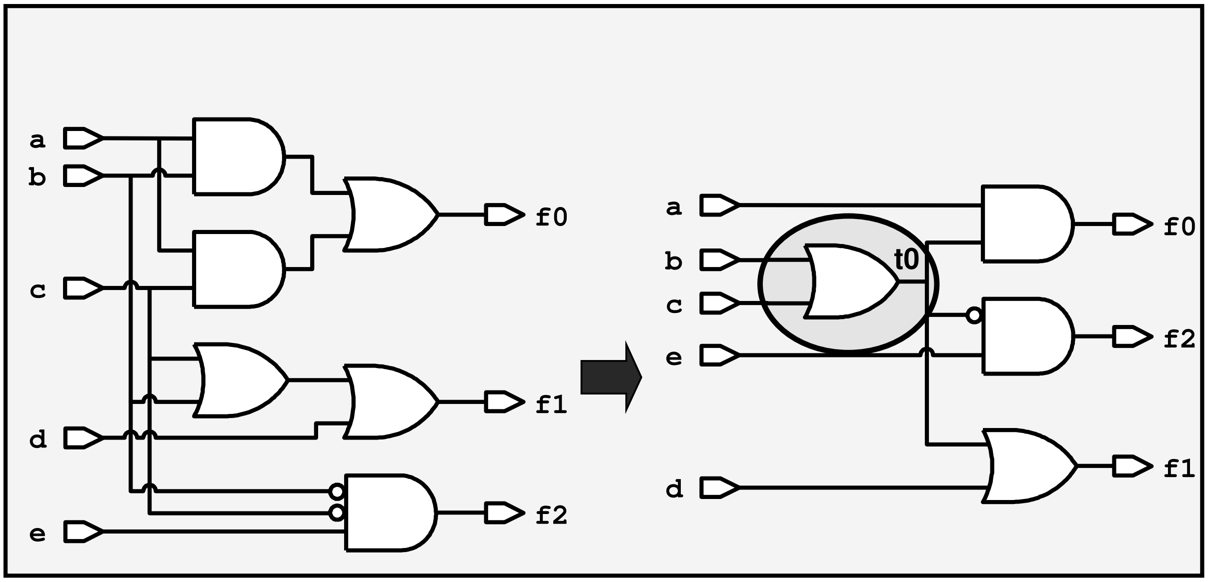
What Is Logic-Level Optimization?

- **After high-level synthesis, circuit function is still represented by GTECH parts**
- **One optimization process occurs by default during logic-level optimization**
  - Structuring
- **Structuring:**
  - Reduces logic using common sub-expressions
  - Is useful for speed as well as area optimization
  - Is constraint-driven

7- 56

Example of Structuring

DC's default logic-level optimization strategy



7- 57

Structuring is a logic optimization step that adds intermediate variables and logic structure to a design. During structuring, Design Compiler searches for sub-functions that can be factored out.

Example:

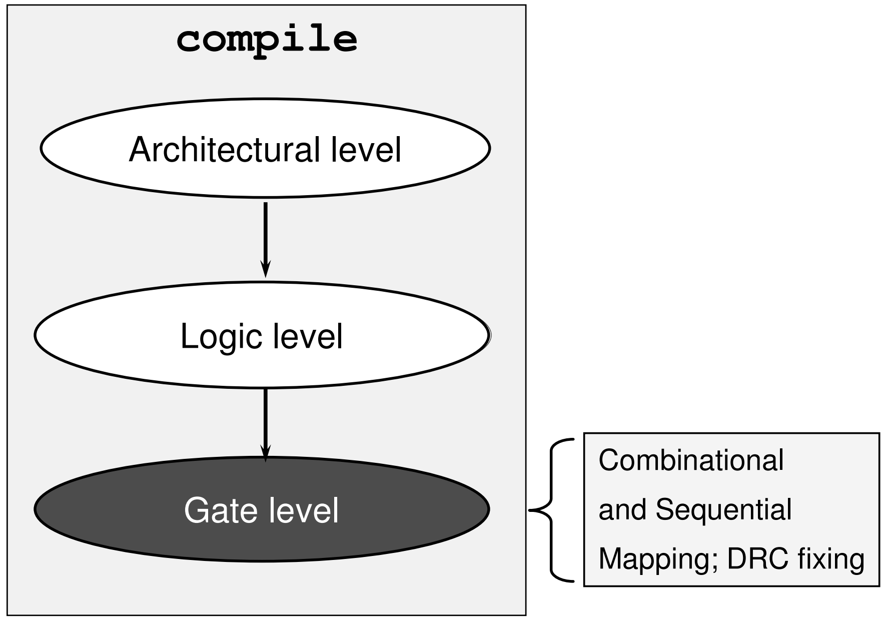
- Before structuring:

```
f0 = a b + a c
f1 = b + c + d
f2 = b' c' e
```

- After structuring:

```
f0 = a t0
f1 = t0 + d
f2 = t0' e
t0 = b + c
```

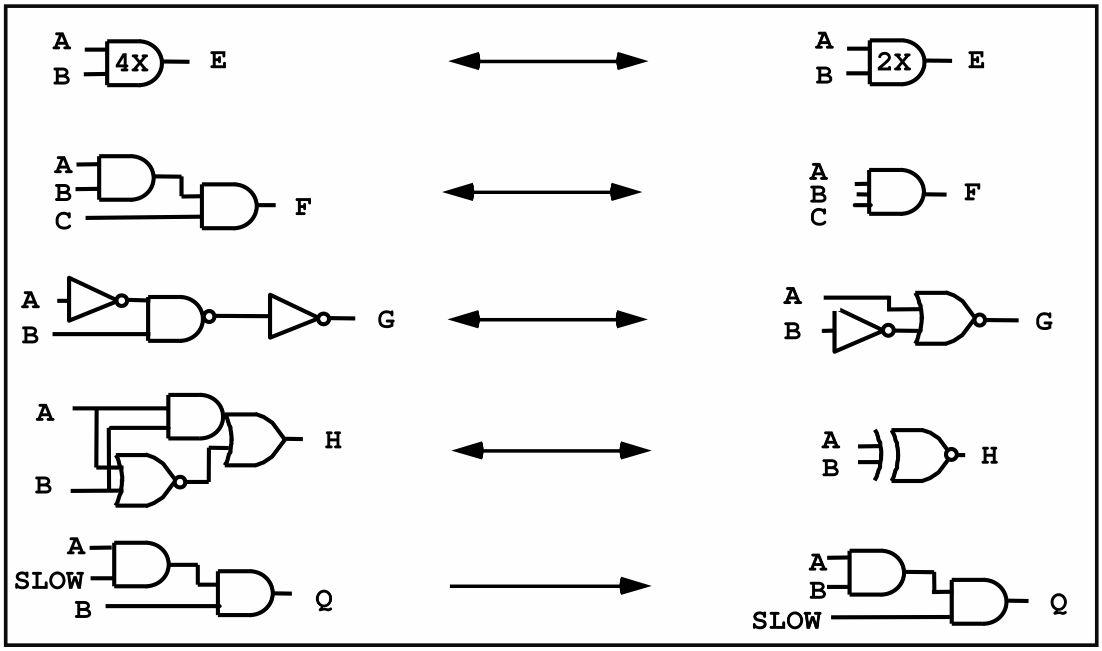
Gate Level Optimization



7-58

Combinational Mapping and Optimization

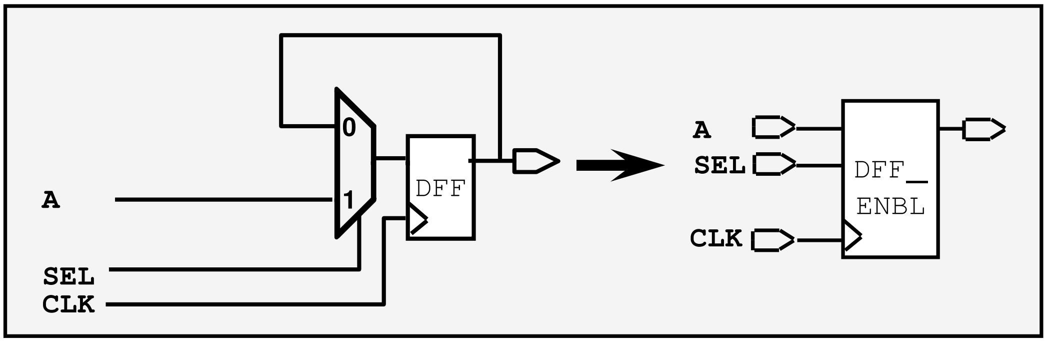
The process of selecting the best combinational logic gates from the target library to generate a design that meets timing and area goals.



7- 59

Sequential Mapping and Optimization

- **The process by which DC maps to sequential cells from the technology library:**
  - Tries to save speed and area by using more complex sequential cells (optimization)



7-60

Default Mapping Optimization Priority

- **Mapping optimization tries to meet the following constraints:**
  - Design rule constraints (DRCs) - Highest priority
  - Timing constraints
  - Area constraint - Lowest priority

DRCs:

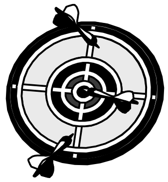
- **Technology libraries contain vendor-specific design rules for each cell, e.g. max_capacitance/transition**
- **DRC fixing entails inserting/removing buffers and re-sizing gates**
  - Has higher priority than delay, by default

7- 61

Appendix 3

Integrated Design-for-Test

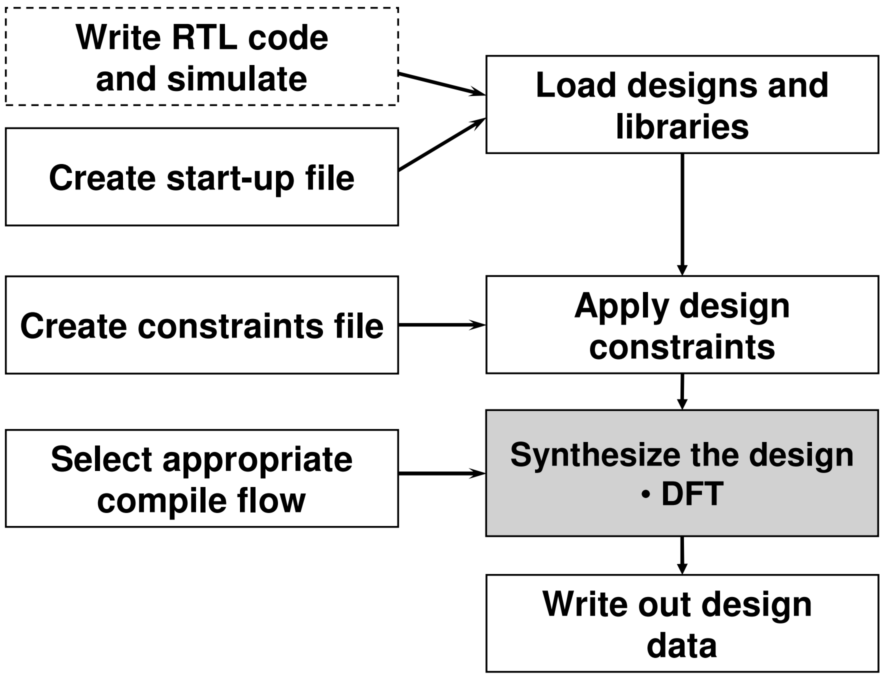
Objectives



- Explain what a stuck-at fault is
- Describe the benefits of scan chains
- Explain what a test-ready compile is and why it is useful
- Recognize the key steps in the DFT flow
- Select the appropriate workshop to learn more about DFT

7- 63

RTL Synthesis Flow



7- 64

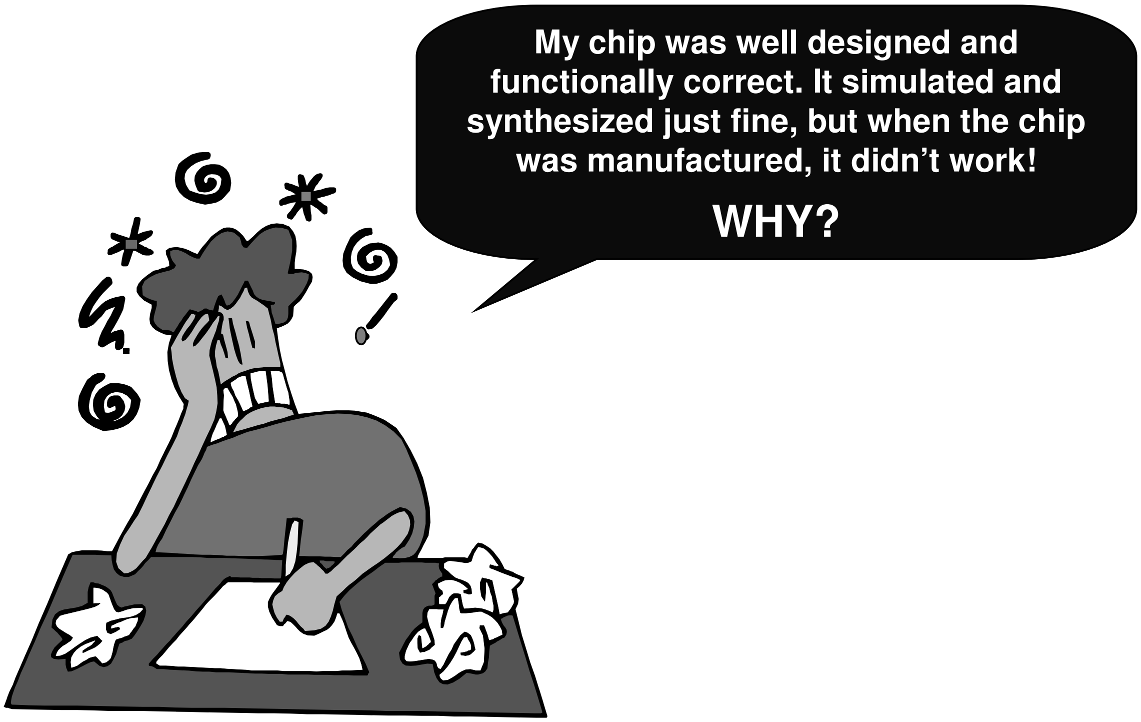
Commands To Be Covered

```
set_scan_configuration -style <multiplexed_flip_flop | \
                    clocked_scan | lssd | aux_clock_lssd>

compile -bound -map high -scan
compile_ultra -retime -timing -scan

source scan_spec.tcl

dft_drc

preview_dft

insert_dft

compile -bound -map high -inc -scan
compile_ultra -inc -scan

dft_drc -coverage_estimate
write_scan_def -out <my_design.def>
```

7-65

We will cover how `set_input_delay` together with `set_driving_cell` affect the data arrival time at input ports.

Chip Defects: They are Not My Fault!



7- 66

Manufacturing Defects

Physical Defects

- Silicon Defects
- Photolithography Defects
- Mask Contamination
- Process Variations
- Defective Oxide

Electrical Effects

- Shorts (Bridging Faults)
- Opens
- Transistor Stuck On/Open
- Resistive Short/Open
- Changes in Threshold Voltage

Logical Effects

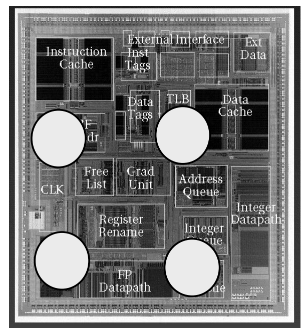
- Logic Stuck-at-0/1
- Slower Transitions (Delay Fault)
- AND-bridging, OR-bridging

7- 67

Physical defects during wafer fabrication (e.g. mask contamination) can cause electrical defects (e.g. shorts, opens), which in turn can create logical defects (e.g. stuck-at-1 or 0).

Why Test for Manufacturing Defects?

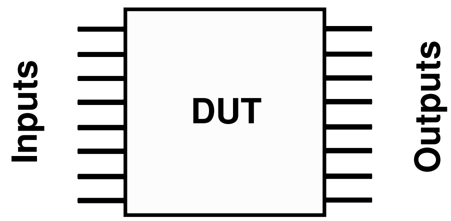
- To detect manufacturing defects and reject those parts before shipment
- To debug the manufacturing process
- To improve process yield



7- 68

How is a Manufacturing Test Performed?

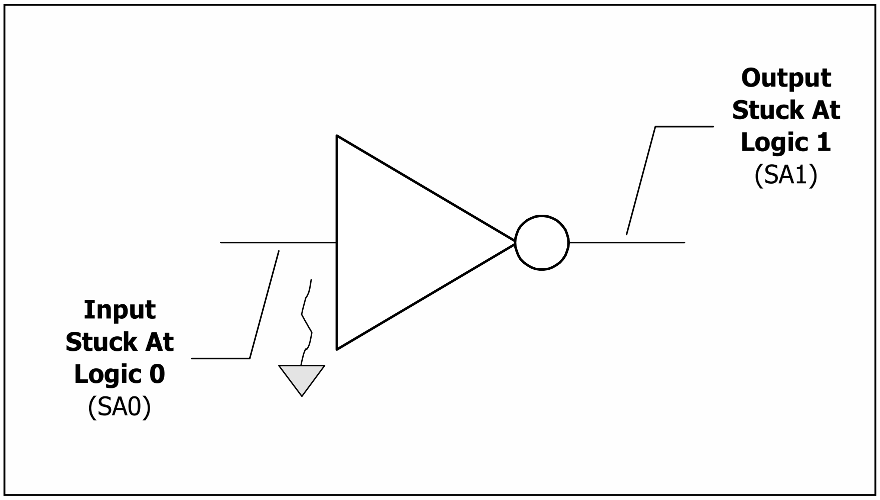
- Automatic Test Equipment (ATE) applies input stimulus to the Device Under Test (DUT) and measures the output response



- If the ATE observes a response different from the expected response, the DUT fails the manufacturing test
- The process of generating the input stimulus and corresponding output response is known as Automated Test Pattern Generation (ATPG)

7- 69

The Stuck-At Fault Model



Stuck-At Fault (SAF): A logical model representing the *effects* of an underlying physical defect.

7- 70

This logical view of an inverter shows the observable effects of underlying defect: A physical defect causes and electrical short on the input, which can be observed as a logical stuck-at-1 on the output.

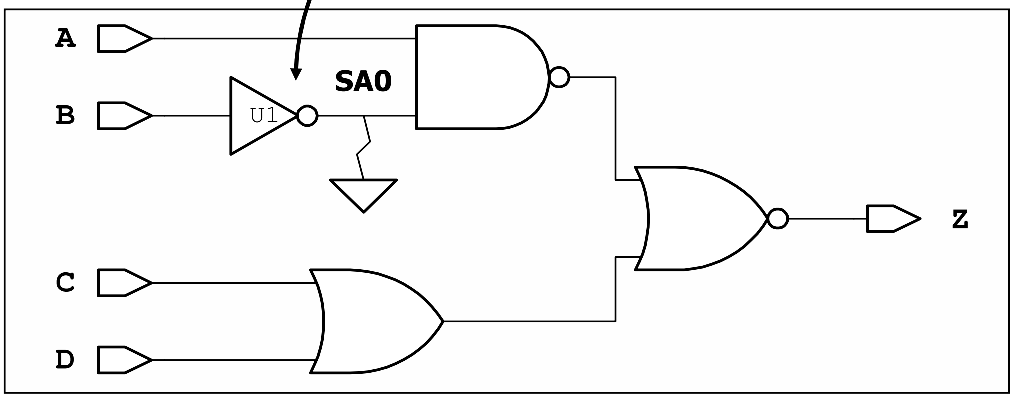
Logical fault models let us focus on observable symptoms—not submicroscopic causes.

The SAF is not the only logical fault model in use, but its benefits make it the most popular.

Algorithm for Detecting a SAF

If this SA0 fault is present then U1/Y stays at logic 0

If not present, then U1/Y is driven to its normal value



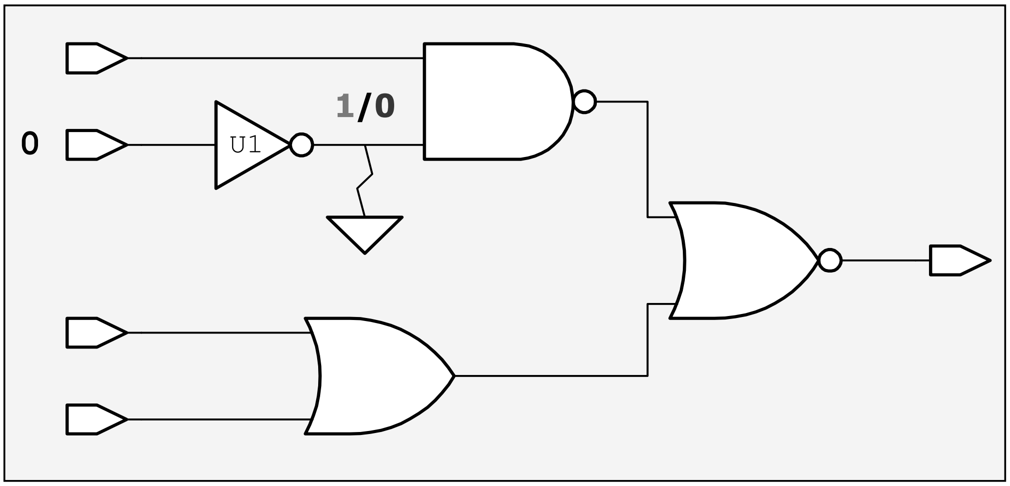
You can exploit this "either/or behavior" to detect the fault.

7- 71

Look at the basic algorithm for detecting a stuck-at fault - the classic D algorithm:
The D algorithm requires no internal probing (consistent with the "rules of the game").
Yet it is able to detect almost any stuck-at fault in a block of combinational logic.
The D algorithm generates a test pattern for one stuck-at fault at a time, either SA0 or SA1.
In its many forms, it is fundamental to all automatic test-pattern generation (ATPG) tools.

Controllability

Is the ability to set internal nodes to a specific value.



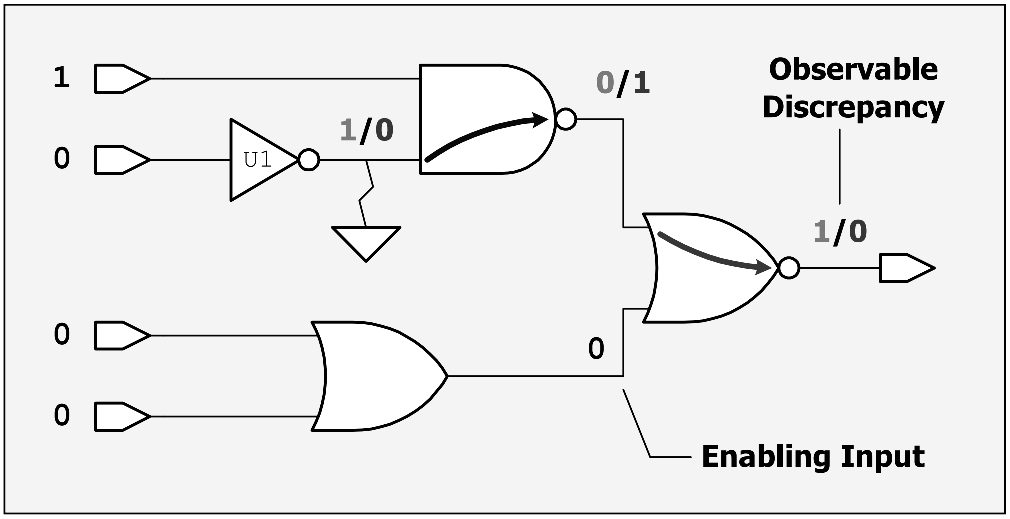
7- 72

1. Pick a specific fault: in this example, the SA0 fault on the output of U1.
2. Generate a detectable error (called a fault effect) by asserting the primary input ports.

If this specific fault is not present on the DUT, then U1/Y is 1. This is the fault-free value.
If the target fault is present on a given DUT, then U1/Y is stuck at 0 (the faulty value).
The key is the measurable difference, or discrepancy, between these two values.

Observability

Is the ability to propagate the fault effect from an internal node to a primary output port.



7- 73

3. Propagate the fault effect to a primary output, in this case, Z.
The fault effect may get inverted, that does not affect the observable discrepancy.
Measure the value at primary output Z to see whether the SA0 fault is present or not.
If you see the expected fault-free response, continue with the rest of the test program.
If you see a faulty response, discard the chip under test as defective.

Fault Coverage

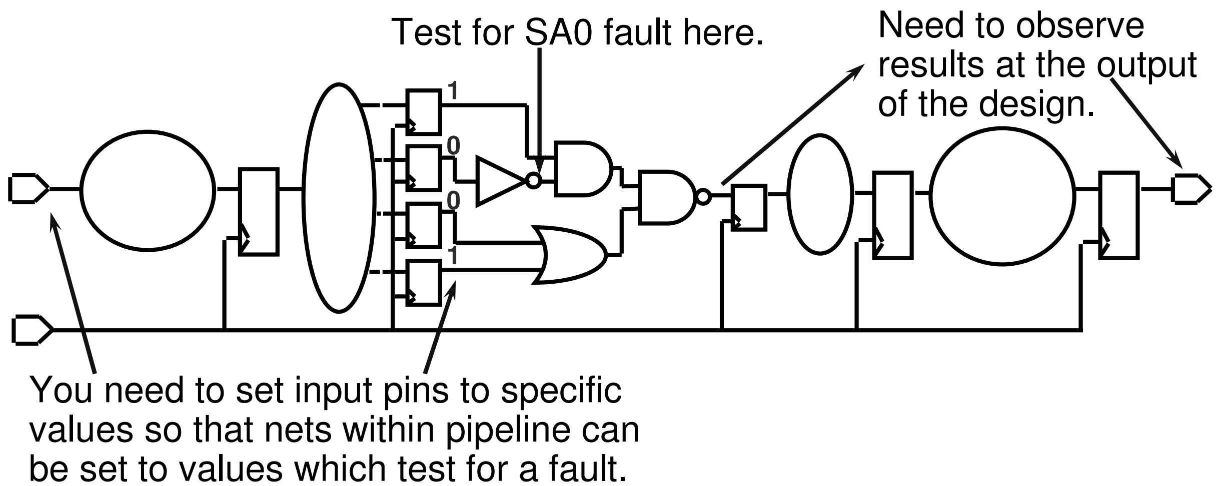
$$\text{Fault coverage} = \frac{\text{number of detectable faults}}{\text{total number of possible faults}}$$

High fault coverage correlates to high defect coverage.

7- 74

Testing a Multistage, Pipelined Design

Testing a multistage, pipelined design can create a lot of complications for you.

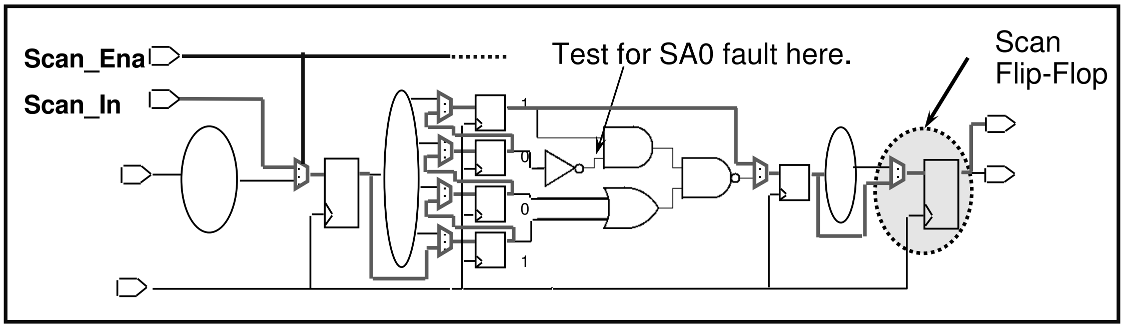


Each fault tested requires a predictive means for both controlling the input and observing the results downstream from the fault.

7- 75

Scan Chains Help

- Inserting a scan chain involves replacing all Flip-Flops with scannable Flip-Flops and serially "stitching" them up
- Scan chains allow data to be easily shifted in to control internal nets (controllability), and shifted out to observe internal faults (observability)

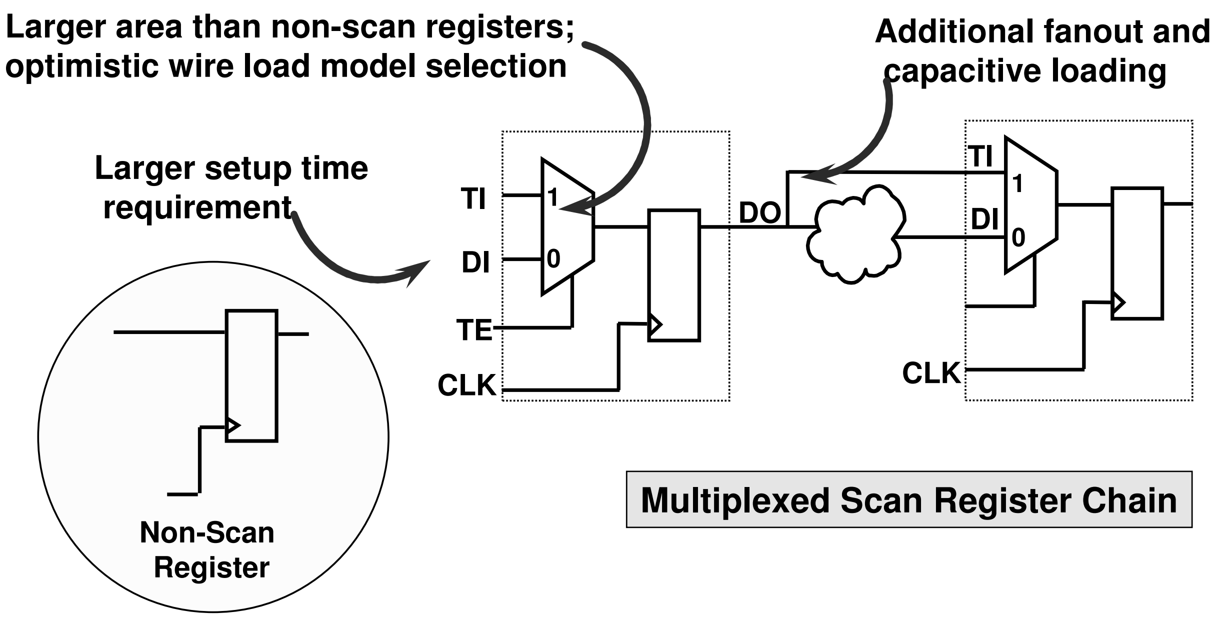


What effect will the mux and scan chain have on circuit timing?

7- 76

Inaccuracy Due to Scan Replacements

If you plan to include internal scan, you must account for the impact of scan registers on a design early in the design cycle.



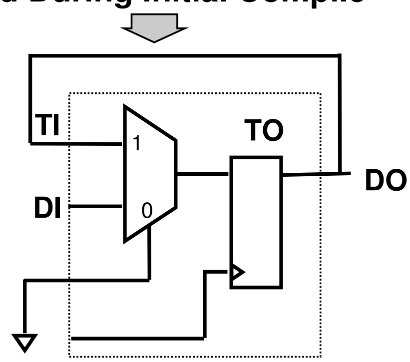
Multiplexed Scan Register Chain

7- 77

Solution: Test-Ready Synthesis

- **Regular registers are replaced with scannable ones, but not chained or "stitched"**
- **Include the scan style in the constraint script file:**
  - `set_scan_configuration -style <multiplexed_flip_flop | clocked_scan | lssd | aux_clock_lssd>`
- **Execute test-ready compile:**
  - `compile -scan` or `compile_ultra -scan`

Scan Register Used During Initial Compile



Benefits

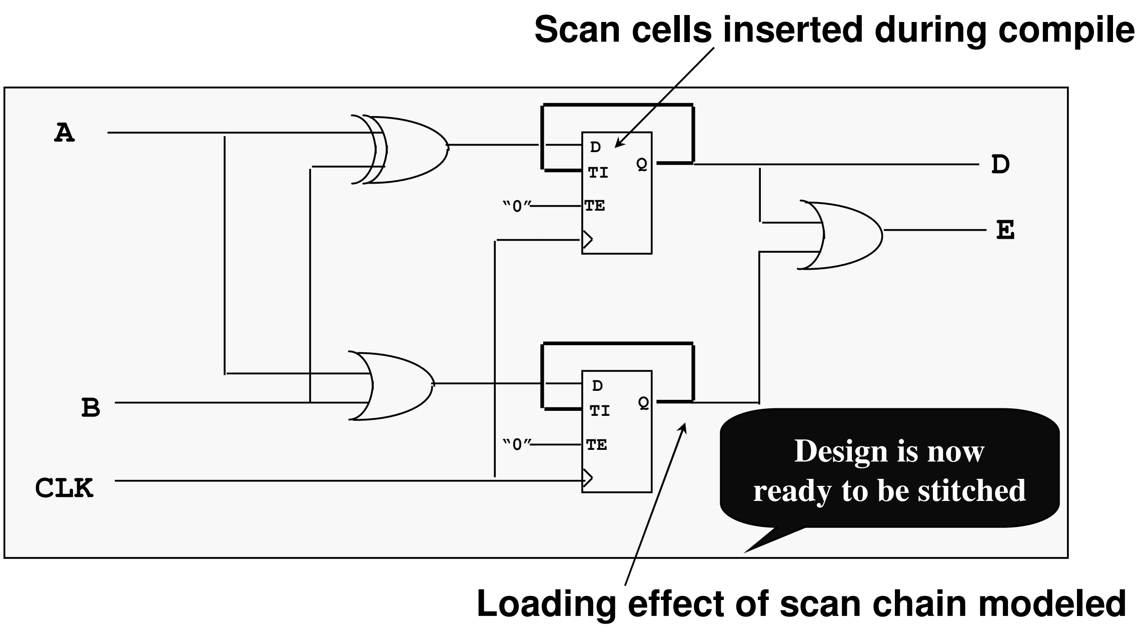
- **Accurate area, timing, and loading modeled up front**
- **Easier synthesis flow -- scan cell insertion performed in one compilation step**

7-78

Test-ready synthesis (compiling using the –scan option) and all following DFT commands requires a DFT Compiler license.

Result of Test-Ready Compile: Example

```
compile ... –scan
```



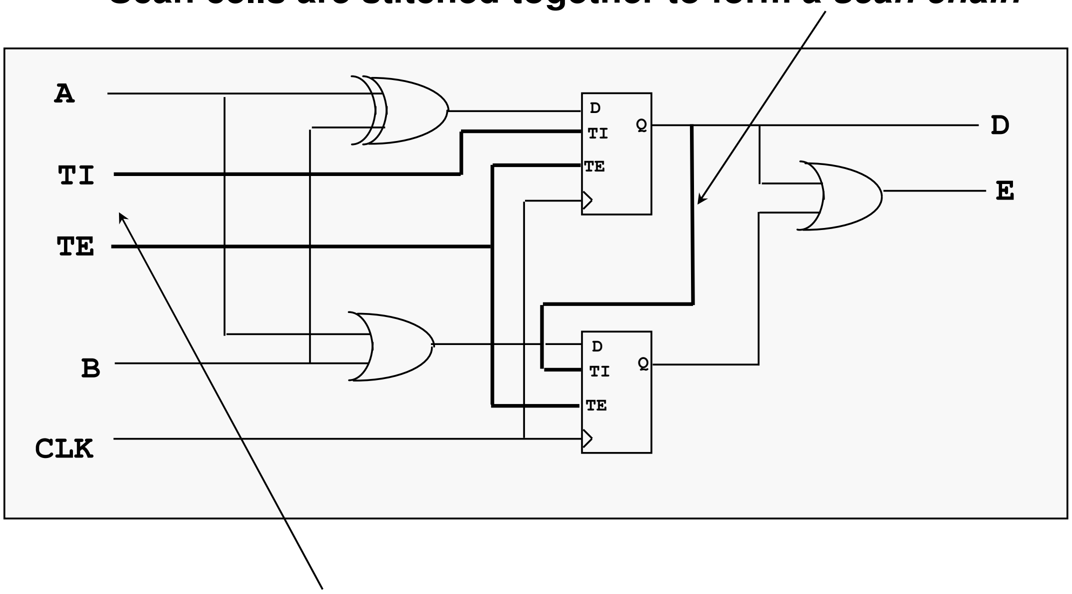
7- 79

Note: The "-scan" option can be used with the *compile* or *compile_ultra* command. Once you perform a test-ready compile (using *–scan*), any subsequent compiles (incremental, or different map efforts or options) MUST also contain the "-scan" option. Otherwise you run the risk of reverting some of the scannable flip-flops back to non-scan flip-flops.

Stitching up the Scan Chain

`insert_dft`

Scan cells are stitched together to form a *scan chain*



Additional test input ports (TE, TI) are created if needed

7- 80

What does `insert_dft` do?

- **Replaces nonscan flip-flops if needed**
- **Creates test access ports (TE, TI) if needed**
- **Stitches access ports and scan cells together**
- **Can add logic to increase fault coverage:**
  - Enforces single-tristate-driver rule
  - Fixes uncontrollable clocks and resets
  - Maintains bidirectional paths during scan-shift cycles
- **Performs an incremental compile to fix timing and DRC violations**
- **With *DFTMAX*: Inserts compression/decompression logic to reduce tester time**
  - Reduces the size of the test patterns for a given number of ports

7- 81

The ordering of the san chain is done alpha-numerically.

Scan specifications must be defined prior to *insert_dft*. This is done through TCL commands in a "scan spec" file. Some of these items include: the number and size of the scan chains; whether or not clock domains can be crossed; scan-in/out/enable port definitions, etc. These scan specifications are covered in the Design for Test workshop.

DFTC will not add fault coverage "fixing logic" by default. You must first enable this behavior with `set_dft_configuration` + `set_autofix_configuration` commands.

Enabling *DFTMAX* requires a separate license and additional commands.

Potential Scan-Shift Issues

DFT Rule of Thumb:
The ATE must **fully** control the clock and asynchronous set and reset lines reaching **all** the scan-path flip-flops.

Design scenarios affecting risk-free scan include:

- **No clock pulse at scan flop due to gating of clock**
- **No clock pulse due to dividing down of the clock**
- **Unexpected asynchronous reset asserted at flop**

7- 82

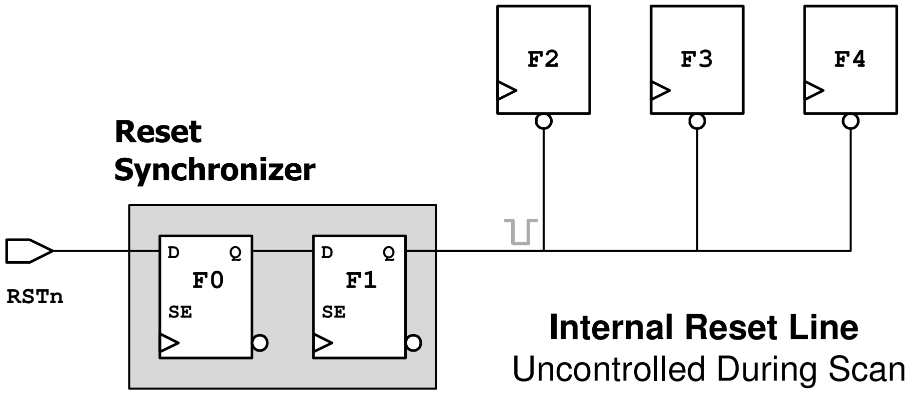
Risk-free scan requires that:

- Every scan flip-flop must receive the synchronizing clock pulses generated by the ATE
- Clock pulses should not be **blocked** from reaching a scan flop due to **gating** of the clock
- Clock edges must arrive regularly at scan flops, without being **divided** or **multiplied**
- No scan flops are unexpectedly **set** or **reset** asynchronously during the scan-shift process

Conclusion:

Run the recommended DRC checks **early and often** to identify and fix most problems.

Example: Unexpected Asynchronous Reset



- **An unexpected reset signal from F1 during scan shifting would over-ride the scan data that you are trying to control or observe, therefore…**
- **All affected flip-flops are excluded from scan chains**
- **This reduces the fault coverage – not good!!**

7- 83

- Risk-free scan requires **inactive** asynchronous set and reset signals during scan-shift.
- This design is a classic DRC violation which allows `F1` to reset flip-flops `F2`, etc.
- Unless the violation is fixed, all affected flip-flops are **excluded** from any scan path.
- If *many* flops are excluded, the impact of this violation is a **major** loss of coverage.
- The DFTC warning for this violation is `TEST-116`, "asynchronous pin uncontrollable."
- Use the "**Auto Fix" feature** as a fast workaround to add the correction logic during scan insertion (DFTC feature).

Report test violations: `dft_drc`

```
dc_shell-t-xg> dft_drc
Warning: Asynchronous pins of cell F2_reg (fdf2a3) are
   uncontrollable. (TEST-116)
Information: The network contains: F2_reg/CLR, rst_int_reg/Q.
   (TEST-281)
Information: There are 479 other cells with the same violation.
   (TEST-171)
...
PROTOCOL VIOLATIONS
    480 Asynchronous pins uncontrollable violations (TEST-116)
...
SEQUENTIAL CELLS WITH VIOLATIONS
   * 480 cells are black boxes
...
```

How may we fix this problem?

7- 84

dc_shell-t> **man TEST-116**

…

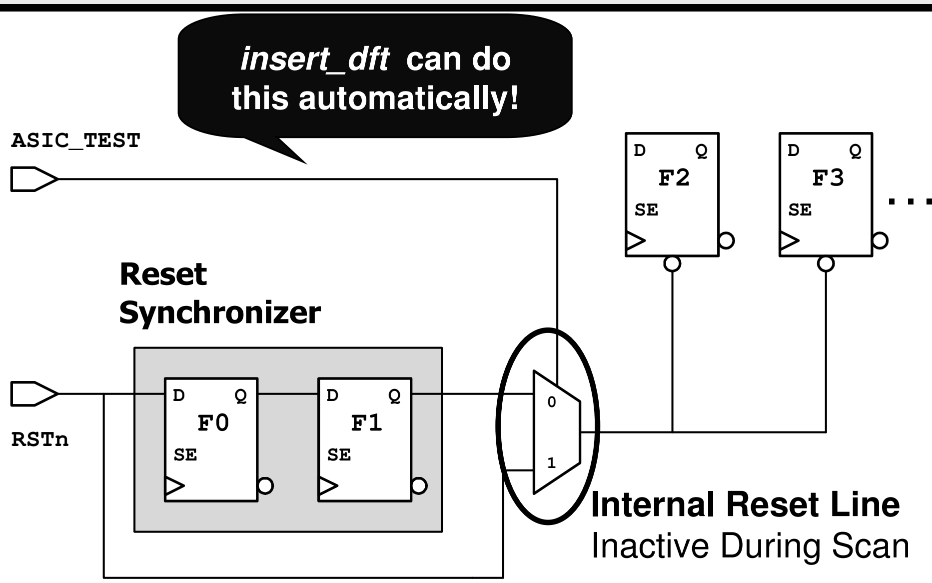
DESCRIPTION

When scan data is shifted in and out of the design-under-test, the asynchronous preset and clear pins of sequential cells must be held in an inactive state. The test design rule checker tries to derive a set of values to apply at the external ports of the design to ensure this condition is met. For the reported cell, no such values could be found to force the asynchronous preset and clear pins to inactive values. This problem usually occurs when the asynchronous preset or clear signal is generated from the state of other sequential devices, as with an asynchronously resetting counter.

WHAT NEXT

To maximize fault coverage, try to make all internal asynchronous preset and clear pins externally controllable. You can do this by introducing additional logic to disable internal preset and clear signals either throughout testing or just during scan shift.

Internal Asynchronous Reset Solution



This solution offers higher coverage — for more DFT effort.

7- 85

- The solution with **optimal coverage** is to multiplex **`RSTn`** around the synchronizer.
- Faults on the internal reset line are thus fully **controllable**—but are they **observable**?
- TetraMAX ATPG makes them observable by using **`RSTn`** in effect as a capture "clock."

- The reset synchronizer module itself is also fully **controllable**—but not **observable**.

- When the autofix feature is turned on DFTC can take care of this issue automatically. This is discussed in the Design for Test Workshop.

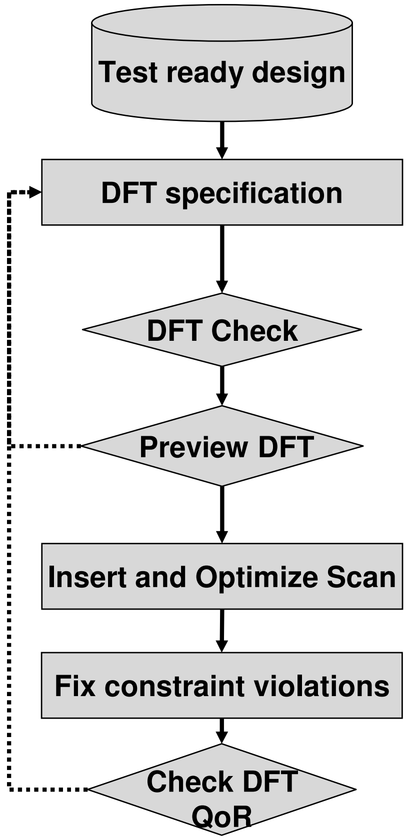
Preview Test Coverage

`dft_drc -coverage` estimates the test coverage using the TetraMAX ATPG engine.

```
dc_shell-t-xg> dft_drc -coverage
 ...
 17378 faults were added to fault list.
 ATPG performed for stuck fault model using internal pattern source.
 ...
 Uncollapsed Stuck Fault Summary Report
 -----------------------------------------------
 fault class                     code   #faults
 ------------------------------  ----  ---------
 Detected                         DT      17352
 Possibly detected                PT          0
 Undetectable                     UD         23
 ATPG untestable                  AU          0
 Not detected                     ND          3
 -----------------------------------------------
 total faults                             17378
 test coverage                           99.98%
 -----------------------------------------------
```

7- 86

DFT Flow




```
set_scan_configuration -style ..
compile -b -map high -scan
compile_ultra -retime -timing -scan
```

```
source scan_spec.tcl
```

```
set_scan_state test_ready
set_dft_configuration...
set_dft_signal. . .
set_scan_path. . .
set_scan_configuration...
create_test_protocol
```

```
dft_drc
```

```
preview_dft
```

All of this, and a lot more, is discussed in the “DFTC 1” workshop!

```
insert_dft
```

```
compile -bound -map high -inc -scan
compile_ultra -inc -scan
```

```
dft_drc -coverage_estimate
write_scan_def -out <my_design.def>
```

7- 87

Test For Understanding

1. **A high fault coverage percentage is bad because this means your design has many faults. T or F?**
2. **The *compile –scan* command does not perform scan chain stitching. T or F?**
3. **DFT Compiler can add correcting logic to fix timing violations. T or F?**
4. ***insert_dft* can fix timing and DRC violations by default. T or F?**

7- 88

1. False. A high fault coverage means that during testing most internal nodes will be checked for faults. Ideally you would like to achieve 100% fault coverage.
2. True. It only inserts scannable flops, but does not stitch them up.
3. False. It can add correcting logic to eliminate DFT design rules, thereby increasing fault coverage.
4. True. It does an incremental compile under the hood.

Design-for-Test Summary

- **Some problems associated with design-for-test can be anticipated and corrected in advance, during the initial *compile* of the RTL code**
- **There is a lot more to Design-for-Test - If you will be running DFTC it is highly recommended that you attend the “Design for Test Workshop”**

7- 89

Summary: Commands Covered

```
set_scan_configuration -style <multiplexed_flip_flop | \
                    clocked_scan | lssd | aux_clock_lssd>

compile -bound -map high -scan
compile_ultra -retime -timing -scan

source scan_spec.tcl

dft_drc

preview_dft

insert_dft

compile -bound -map high -inc -scan
compile_ultra -inc -scan

dft_drc -coverage_estimate
write_scan_def -out <my_design.def>
```

7- 90

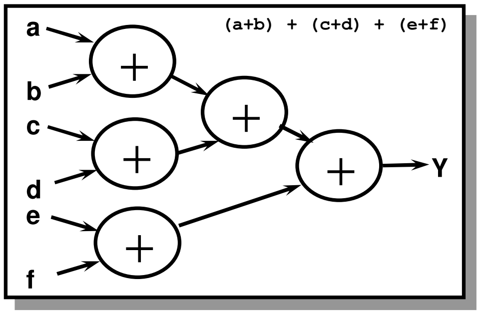
We will cover how `set_input_delay` together with `set_driving_cell` affect the data arrival time at input ports.

Appendix 4

Examples of macro architecture optimization for arithmetic operations

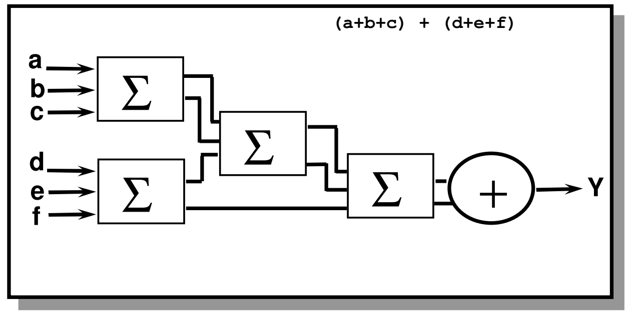
The Problem: Designing Optimal Arithmetic

Conventional Arithmetic



- Additions in groups of 2
- Complete computation at the end of each operator

Carry-Save-Addition (CSA) Arithmetic



- Additions in groups of 3
- Partial intermediate computation (saves area and improves speed)

Well understood by designers and automatically performed by synthesis tools today

Design tricks used by advanced designers; but requires pages of RTL code to achieve optimum results

7- 92

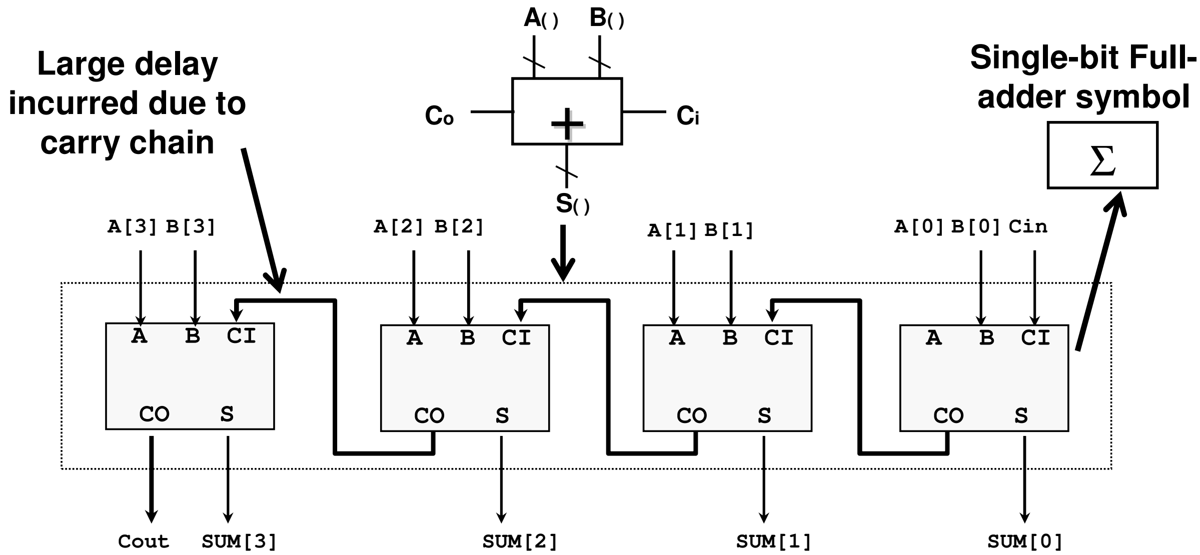
CSA is an adder architecture used to calculate intermediate partial sum values.

CSA architecture saves the entire carry vector to avoid carry-propagation delay overhead.

A tree structure of CSAs is typically used to reduce the number of input vectors from N down to 2.

A conventional adder (ripple, cla, clf) is required at the bottom of the CSA tree to add the final 2 vectors and produce the final sum.

4-bit Ripple Adder: Small but Slow



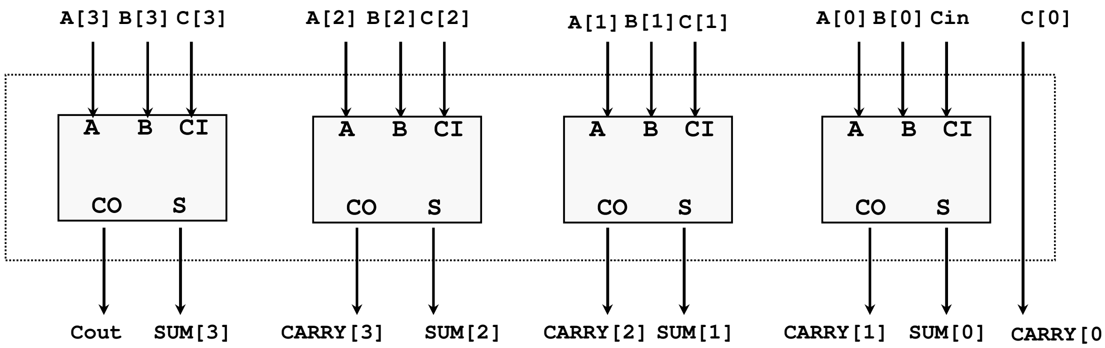
- A 4-bit ripple adder uses only 4 *full adders*, which results in the smallest possible adder logic
- However, due to the carry chain rippling from LSB to MSB, it is also the slowest: Delay = 4 x Full-adder delay

7- 93

This is a four-bit ripple adder.
Note that there is a carry chain rippling from l.s.b. to m.s.b.

Carry Save Adder Building Block

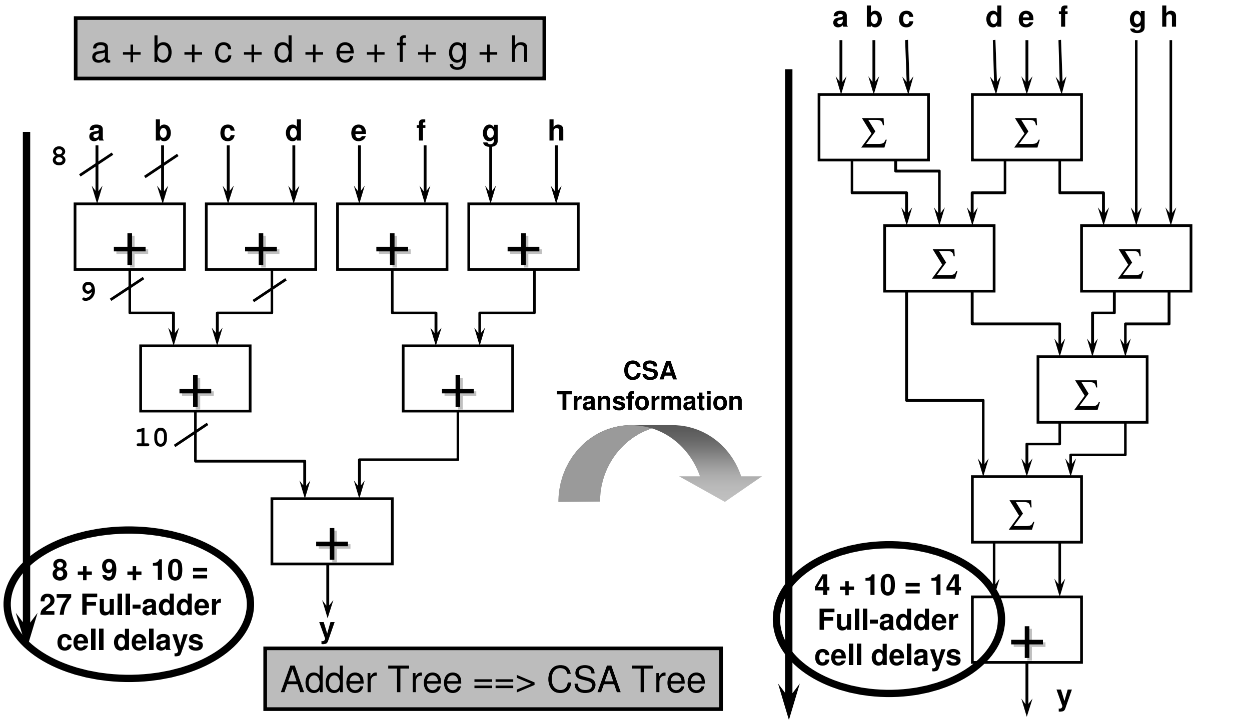
4-bit CSA Operator (CSA = Carry Save Adder)



- There is no carry chain rippling from L.S.B. To M.S.B.
- Produces an intermediate or *partial sum* result
- Each CO pin goes to the next CSA in the arithmetic chain (not shown here)
- The full sum is computed at the last stage (also not shown here)
- Same small area as a ripple-adder, but only 1 Full-adder delay

7- 94

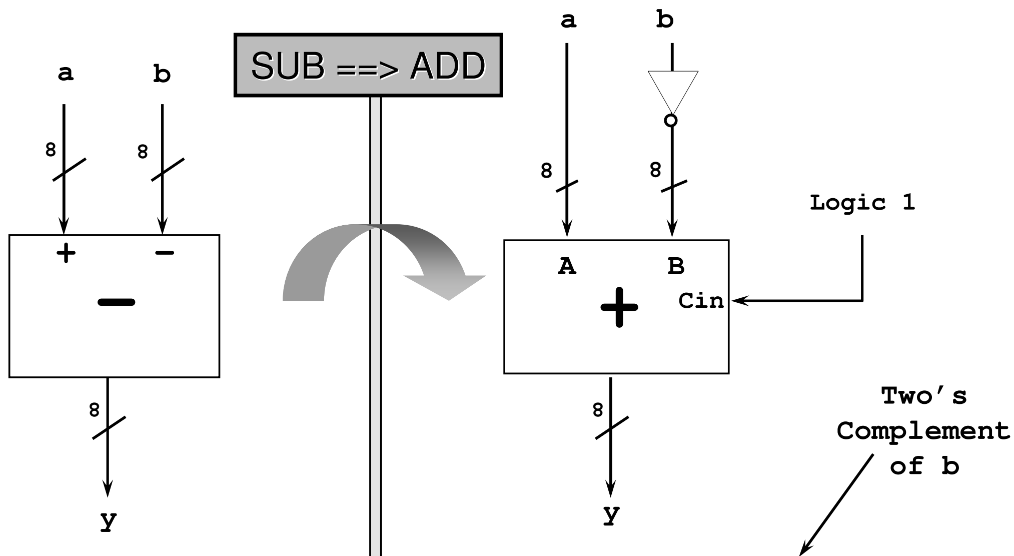
CSA Tree Example



A CSA tree is much faster than a ripple-adder tree, with the same small area

7- 95

Converting *Subtractors* into *Adders*



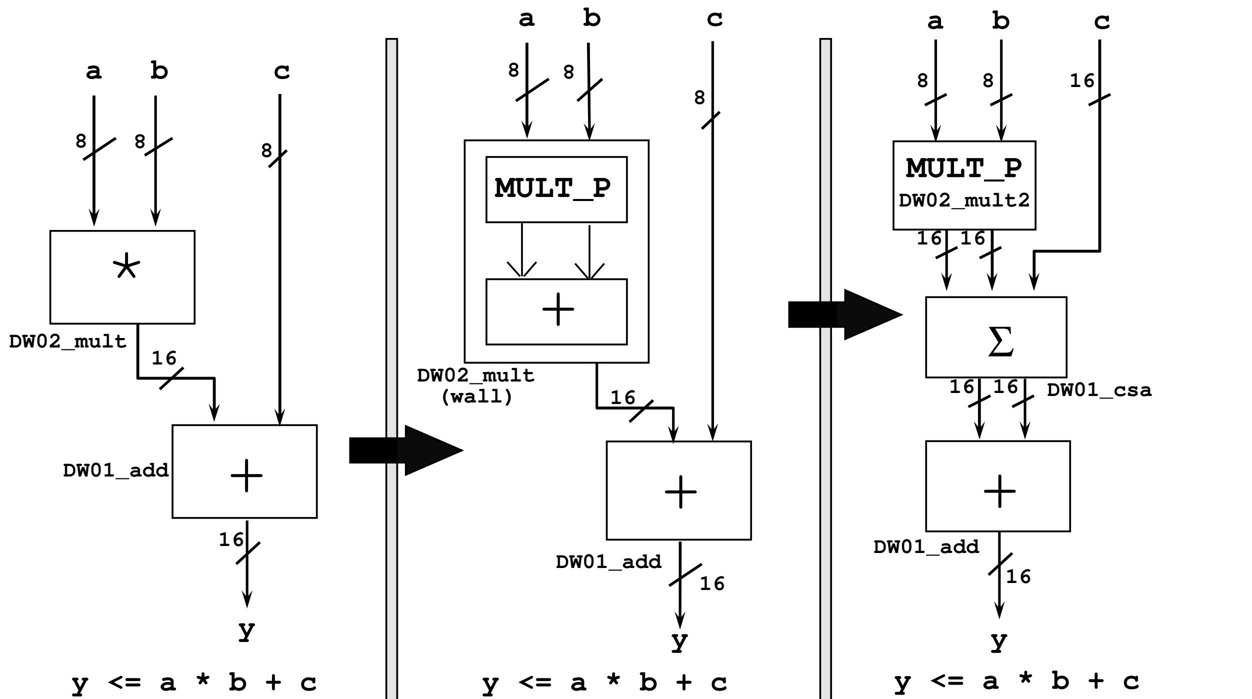
y <= a - b

y <= a + (~b) + 1

A CSA tree requires adders, so subtractors are first converted into adders

7- 96

Transforming *Multipliers* with CSA Adders



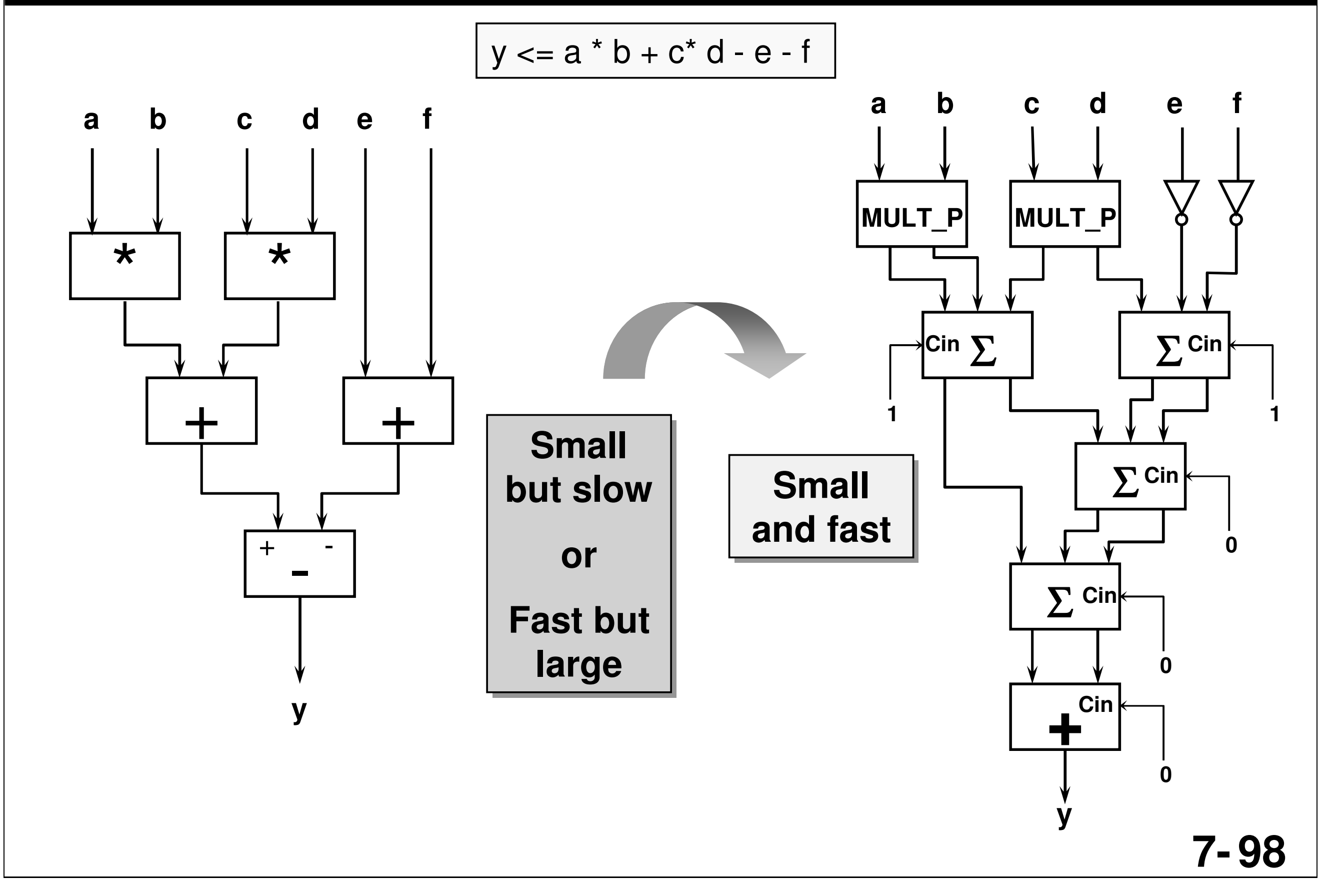
Original Implementation

Split the multiplier into a partial multiplier and adder

Transform and merge the spun-off adder with another adder

7- 97

Example: Combining all Transformations



Note: Order of Summations.

Subtraction by e becomes addition with its one's complement plus 1.
Subtraction by f becomes addition with its one's complement plus 1.
~e and ~f take priority to be summed since they should arrive earlier than the MULT_P outputs.
Two's complement inversion of e and f generates two '1's. Adding these two '1's costs zero area since the addition is accomplished by connecting two Cin inputs to Logic1.

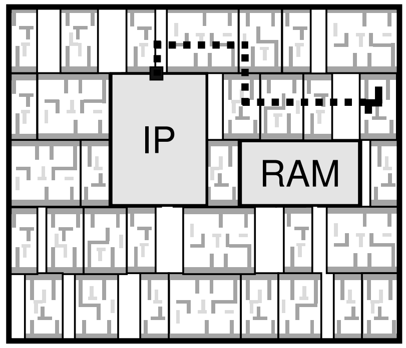
Unused Cin inputs are connected to Logic0.

Appendix 5

Milkyway Reference and Design Libraries

Milkyway Reference and Design Libraries

- **The `compile_ultra` command in *Topographical mode* performs under-the-hood placement to estimate interconnect RCs**

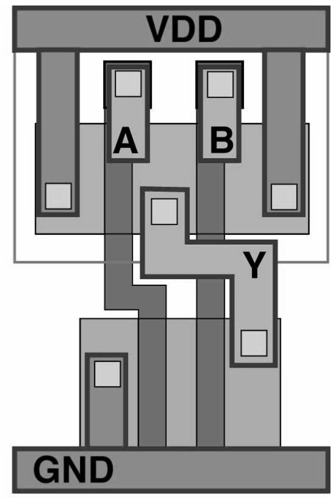


- ***Topographical mode* requires "physical" libraries - standard cell and IP/macro libraries**
  - Specified as *Milkyway reference libraries*

7- 100

What is a Standard Cell Library?

- A Standard Cell is a predesigned layout of one specific basic logic gate
- Each cell usually has the same standard height
- A Standard Cell Library contains a varied collection of standard cells
- Libraries are usually supplied by an ASIC vendor or library group



Layout View

2-Input NAND Gate

7- 101

Standard cell libraries describe the layout of basic combinational and sequential logic gates like:

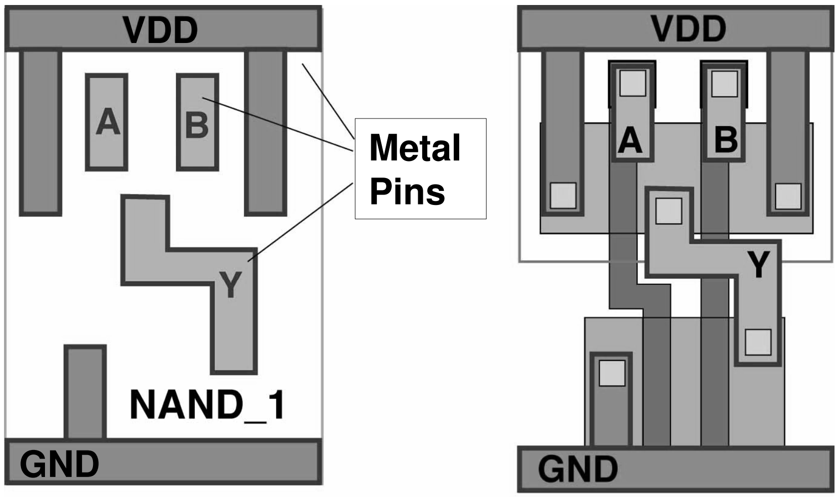
- inverters
- buffers
- ands
- ors
- nands
- nors
- muxes
- latches
- flip-flops

Standard cell libraries are generally called "reference" libraries. These reference libraries are technology specific and are usually created and provided by an external ASIC vendor or internal library group.

In addition to standard cell libraries, reference libraries also contain pad cell libraries for signal IO and power/ground pad cells, as well as macro cell or IP libraries for reusable IP like RAMs, ROMs, and other predesigned, standard, complex blocks.

"Layout" vs. "Abstract" Views

- **A standard cell library also contains a corresponding *abstract* view for each layout view**
- **Abstract views contain only the minimal data needed for Place & Route**



Abstract View (Milkyway FRAM view) **Layout View (Milkyway CEL view)**

7- 102

The use of abstract views instead of cell views enables smaller, more efficient design databases to be built and manipulated by P&R tools.

The abstract views do not contain underlying device details (poly, diffusion, n-wells, contacts). They contain only the following:

- **Outline of the cell** (The "placement" tool places each cell in available sites in the core of the chip. All the placer needs to know is the size and shape (outline) of the cell.)

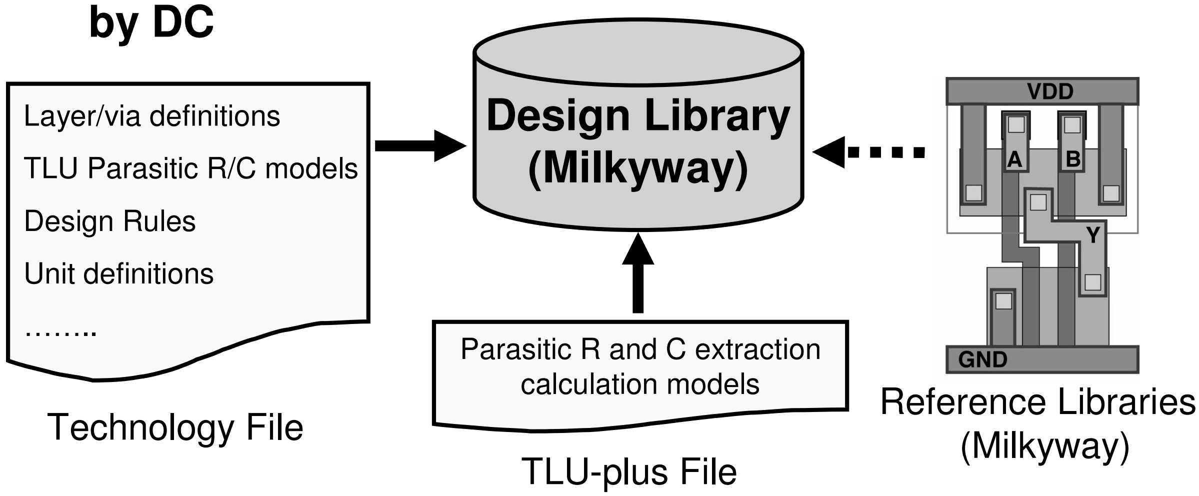
- **Pin locations and layer** (Pins are usually metal connections to the underlying poly or diffusion areas of the devices. The "router" tool uses these pin locations to route metal wires or connections to each cell)

- **Metal blockages** (Areas in a cell where metal of a certain layer can not be routed because that layer is being used in the full layout of the cell, but does not show up as a pin in the abstract view. The example layout above is too simple to require blockages. These are more commonly needed in macro cells)

The layout and abstract views of all standard cells are built by a Library group. These cells are usually initially built in a standard format called GDSII and then converted into tool-specific formats (e.g. Milkyway database format for use by Synopsys tools).

Milkyway Reference and Design Libraries

- **The user must also create a *Milkyway design library* used to "package up" the reference libraries, along with a technology file and TLU-plus models for use by DC**



- ***Target* and *link libraries* must still be specified for logical, timing and power info**

7- 103

The reference libraries (standard cell and, if applicable, macro cell or IP libraries), along with the technology file and TLU-plus file (if applicable) are all provided by the back-end vendor or library group.

The arrow from the reference libraries is dashed to indicate that the reference libraries are not included in the design library – instead, the design library contains a "pointer" to the libraries.

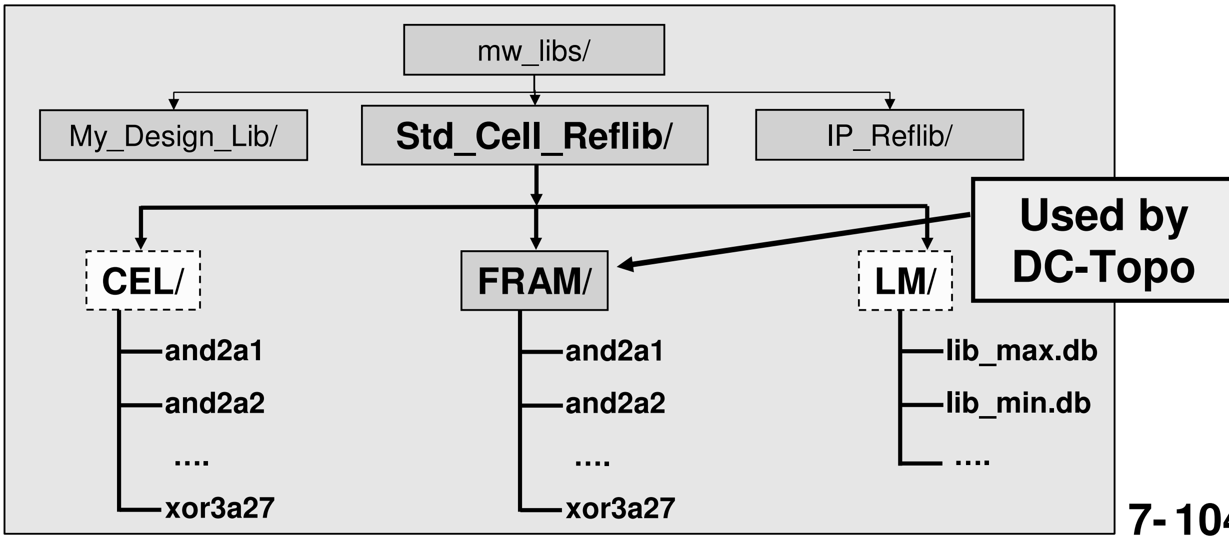
Note: The recommended format for transferring design data between Design Compiler and Physical Compiler or IC Compiler is ddc – not Milkyway.
For DC-to-Astro, output a Verilog netlist, along with the sdc constraints file.

UNIX Structure of a Milkyway Libraries

Reference **libraries are** ***Milkyway*** **databases, with a similar UNIX structure as a** ***design*** **library database.**

Reference libraries contain different ***views*****:**

- CEL: The full layout view
- FRAM: The abstract view used for P&R, as well as DC Topographical
- LM: Logic Model with Timing and Power info, used by layout tools



7- 104

The timing and power information of the LM views is not used by DC Topographical – it gets its timing (and if applicable, power) information from the target and link libraries.

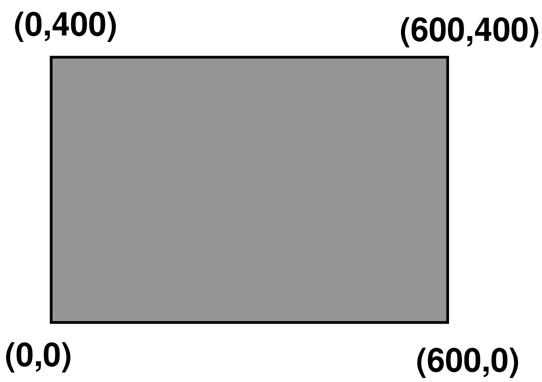
The full layout (CEL) view is used after Placement and Routing (P & R) are completed, and the design is ready to be "taped out". This means that the full layout representation of the design is written out in a standard format called "GDSII" – this file must contain ALL the physical data of the design, not just the "abstract" data used during P & R.

Appendix 6

Additional Physical Constraints examples

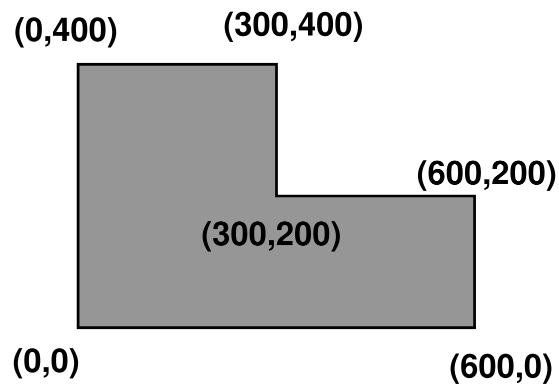
Defining Exact Core Area

Defining a Rectangular Core Area




```
set_placement_area \
 -coordinate {0 0 600 400}
```

Defining a Rectilinear Core Area

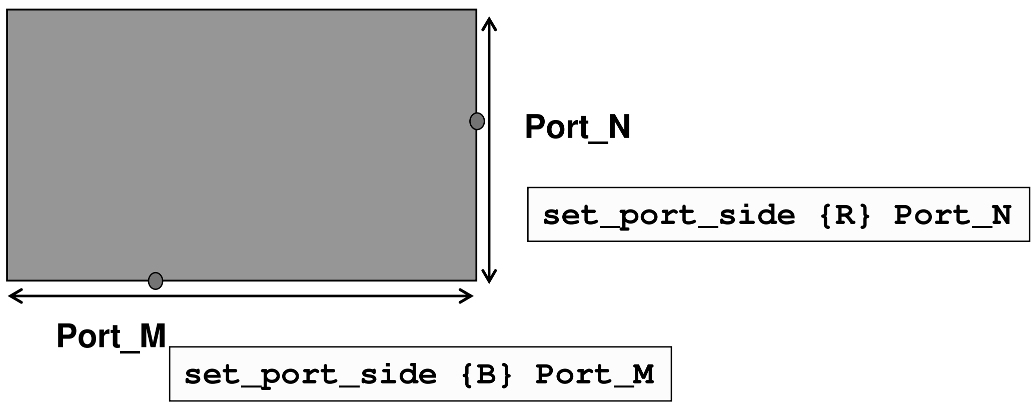



```
set_rectilinear_outline \
         -coordinate    \
 {0 0 600 0 600 200 300 \
  200  300 400 0 400}
```

7- 106

Defining Relative Port Sides

- **Port side dictates along which side a port will be placed**
  - Valid sides are left (L), right (R) top (T) or bottom (B)
  - The port can be placed at any location along the specified side

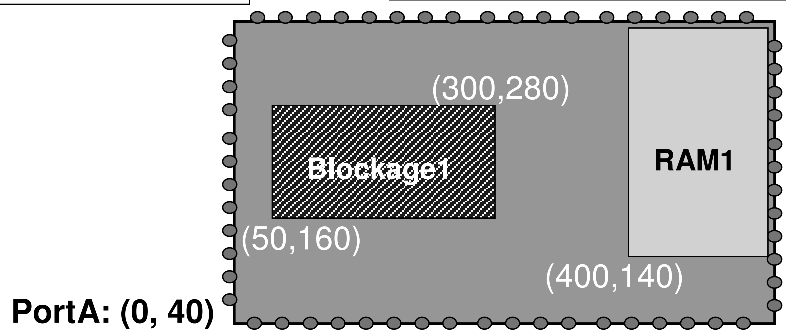


7- 107

Defining Exact Ports, Macros and Blockages

```
create_placement_keepout \
     -name Blockage1 \
     -coordinate  \
       {50 160  300 280}
```

```
set_cell_location \
 -coordinate {400 140} \
 -orientation {N} -fixed \
    RAM1
```




```
set_port_location \
 -coordinate {0 40} PortA
```

7-108

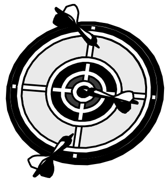
Agenda

DAY 2

5	Partitioning for Synthesis
6	Environmental Attributes
7	Compile Commands
8	Timing Analysis
9	More Constraint Considerations

Synopsys 10-I-011-SSG-013 8- 1

Unit Objectives

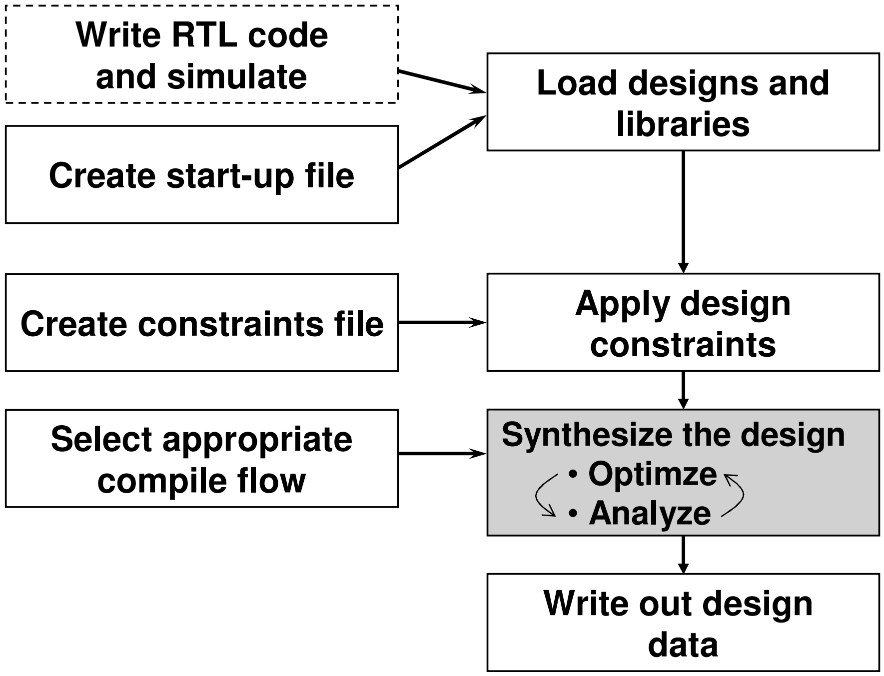


After completing this unit you should be able to:

- **Generate timing reports, with additional command options as needed, to diagnose timing constraint violations**

8- 2

RTL Synthesis Flow



8- 3

Commands Covered in this Unit

```
report_timing
                  [ -delay max | min ]
                  [ -to pin_port_clock_list ]
                  [ -from pin_port_clock_list ]
                  [ -through pin_port_list ]
                  [ -group]
                  [ -input_pins ]
                  [ -max_paths path_count ]
                  [ -nworst paths_per_endpoint_count ]
                  [ -nets ]
                  [ -capacitance ]
                  [ -significant_digits number]
```

8- 4

Timing Reports

The `report_timing` command:

See Appendix
for more details

- **Breaks the design down into individual timing paths**
- **Analyzes each timing path at least twice for single-cycle max-delay timing**
  - Rising endpoint and falling endpoint
- **Generates a default, four-section report which includes:**
  - One path, the worst violator[1], per path group[2]
  - Maximum delay or *setup* timing only
    - No *hold* timing
    - No DRC
    - No area

8- 5

[1] The worst violator is defined as the path with the largest or *worst negative slack* (WNS). If a clock group has no violating paths then the path with the smallest positive slack is reported.

[2] A path group is a group of timing paths which are all captured by the same clock, by default. In a later Unit user-defined path groups will be discussed.

Timing Report: Path Information Section

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : TT
Version: 2007.03
Date   : Tue Mar 20 16:48:52 2007
****************************************

Operating Conditions: slow_125_1.62   Library: ssc_core_slow
Wire Load Model Mode: enclosed

  Startpoint: data1 (input port clocked by clk)
  Endpoint: u4 (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  TT                 5KGATES               ssc_core_slow
```

8- 6

Timing Report: Path Delay Section

Individual Contribution to Path Delay

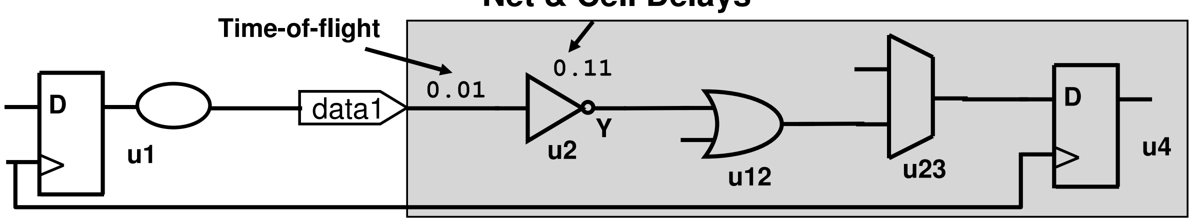
Running Total of the Path Delay

```
Point                                Incr       Path
---------------------------------------------------------
clock clk (rise edge)                0.00       0.00
clock network delay (ideal)          0.50       0.50
input external delay                 1.00       1.50 f
data1 (in)                           0.04       1.54 f
u2/Y (inv1a1)                        0.12       1.66 r
u12/Y (or2a1)                        0.26       1.92 r
u23/Y (mx2d2)                        0.23       2.15 f
u4/D (fdef1a1)                       0.00       2.15 f
data arrival time                               2.15
```

Signal Transition

Arrival Time

Net & Cell Delays



8- 7

By default, the data arrival time at an input port is equal to the `set_clock_latency` (network + source latency) of the input launching clock (clock network delay), plus the `set_input_delay` (input external delay), plus the delay associated with the input transition or `set_driving_cell` (reported as an incremental delay on the input port itself.

If the `set_input_delay` includes the `-network_latency_included` and/or `-source_latency_included` options then DC assumes that the launching clock's latency is already included on the `set_input_delay` number, so no additional latency is added on to the input data arrival time.

The incremental number shown in the timing report on the output pin of a cell is the sum of the cell delay and the time-of-flight of the net on the input pin of the cell. To see the net delay separately from the cell delay, include `-input_pins`, and increase the significant digits in the timing report with `-significant_digits <#>`.

Timing Report: Path Required Section

Clock Edge

```
Point                                  Incr      Path
-----------------------------------------------------
clock clk (rise edge)                  2.00      2.00
clock network delay (ideal)            0.50      2.50
clock uncertainty                     -0.27      2.23
U4/CLK (fdef1a1)                       0.00      2.23
library setup time                    -0.06      2.17
data required time                               2.17
```

From the Library

Data must be valid by this time

8- 8

The incremental

Timing Report: Summary Section

```
-------------------------------------------------
data required time                         2.17
data arrival time                         -2.15
-------------------------------------------------
slack (MET)                                0.02
```

Either **(MET)** or **(VIOLATED)**

Timing margin or *slack*:
(positive number = met
negative number = violation)

8- 9

Timing Report: Options

The default behavior of `report_timing` is to report the path with the worst slack within each path group.

```
report_timing
             [ -delay max/min ]
             [ -to pin_port_clock_list ]
             [ -from pin_port_clock_list ]
             [ -through pin_port_list ]
             [ -group]
             [ -input_pins ]
             [ -max_paths path_count ]
             [ -nworst paths_per_endpoint_count ]
             [ -nets ]
             [ -capacitance ]
             [ -significant_digits number ]

             ...
```

Can you guess which option reports cell and net delays separately?

8- 10

`report_timing` **`-input_pins -significant_digits 6`**

This option shows the delay of the net connected to the input pin.

```
 Point                                             Incr        Path
---------------------------------------------------------------------------
 I_PRGRM_DECODE/Crnt_Instrn[25] (PRGRM_DECODE) 0.000000    2.040000 f
 I_PRGRM_DECODE/U362/A (buf1a2)                0.000246    2.040246 f
 I_PRGRM_DECODE/U362/Y (buf1a2)                0.480000    2.530246 f
```

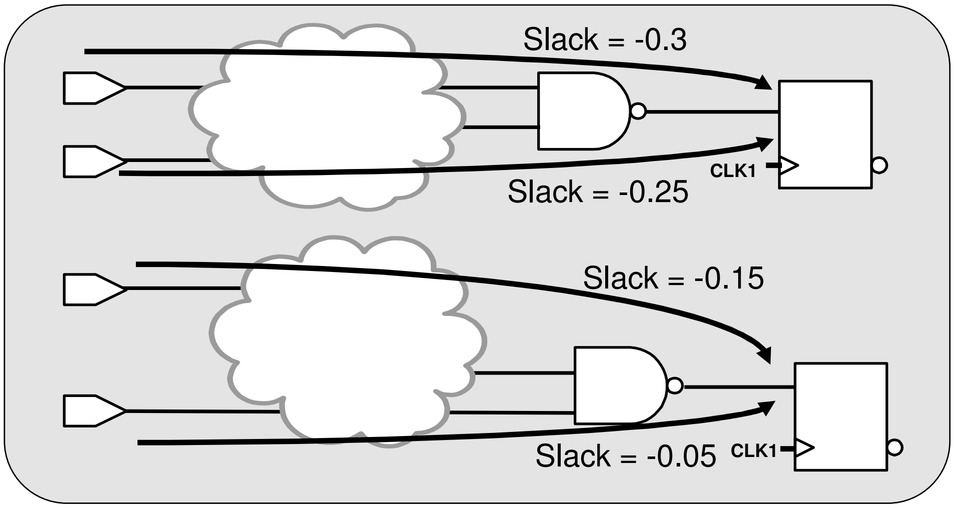
`report_timing` **`-nets`**

The option shows the net name and the fanout, it does not report the delay.

```
 Point                                          Fanout   Incr       Path
---------------------------------------------------------------------------
 I_PRGRM_DECODE/Crnt_Instrn[25] (PRGRM_DECODE)           0.00       2.04 f
 I_PRGRM_DECODE/Crnt_Instrn[25] (net)                    0.00       2.04 f
 I_PRGRM_DECODE/U362/Y (buf1a2)                          0.48       2.53 f
 I_PRGRM_DECODE/n984 (net)                         4     0.00       2.53 f
 I_PRGRM_DECODE/U371/Y (xor2a0)                          0.56       3.09 f
```

ANSWER: `-input_pins`! By default, the delays are reported to cell output pins; this option requests that the delays to cell input pins are also included. The timing arc from a cell's output pin to the next cell's input pin represents the net delay.

Example –nworst vs. –max_paths



What is the *WNS* in this example?

Which paths are reported with `report_timing`?

Which paths will be reported with `report_timing –max_paths 2`?

What changes with `report_timing –nworst 2 –max_paths 2`?

8- 11

Since all 4 paths are part of the same path group, only one path, the path with the worst negative slack (WNS) of -0.3 is reported with a default `report_timing`.

`report_timing -max_paths 2` will generate two worst slack reports, but only allowing one path per endpoint. In this example it will give you the paths with Slack = -0.3 and Slack = -0.15.

`report_timing -max_paths 2 -nworst 2` will generate two worst slack reports, but this time allowing up to two paths per endpoint. In this example it will give you the paths with Slack = -0.3 and Slack = -0.25.

Timing Analysis Exercise

```
Point                                         Incr       Path
clock clk (rise edge)                         0.00       0.00
clock network delay (ideal)                   0.80       0.80
input external delay                          8.40       9.20 f
addr31 (in)                                   0.04       9.24 f
u_proc/address31 (proc)                       0.00       9.24 f
u_proc/u_dcl/int_add[7] (dcl)                 0.00       9.24 f
u_proc/u_dcl/U159/Q (NAND3H)                  0.62       9.86 r
u_proc/u_dcl/U160/Q (NOR3F)                   0.75      10.61 f
u_proc/u_dcl/U186/Q (AND3F)                   0.33      10.94 f
u_proc/u_dcl/U135/Q (NOR3B)                   0.79      11.73 r
u_proc/u_dcl/ctl_rs_N (dcl)                   0.00      11.73 r
u_proc/u_ctl/ctl_rs_N (ctl)                   0.00      11.73 r
u_proc/u_ctl/U126/Q (NOR3B)                   1.34      13.07 f
u_proc/u_ctl/U120/Q (NAND2B)                  0.48      13.55 r
u_proc/u_ctl/U122/Q (OR2B)                    0.68      14.23 r
u_proc/u_ctl/read_int_N (ctl)                 0.00      14.23 r
u_proc/int_cs (proc)                          0.00      14.23 r
u_int/readN (int)                             0.00      14.23 r
u_int/U17/Q (INVB)                            0.11      14.34 f
u_int/U16/Q (AOI21F)                          1.28      15.62 r
u_int/U60/Q (AOI22B)                          1.44      17.06 f
u_int/U68/Q (INVB)                            0.17      17.23 r
u_int/int_flop_0/D (DFF)                      0.03      17.26 r
data arrival time                                       17.26

clock clk (rise edge)                        12.50      12.50
clock network delay (ideal)                   0.80      13.30
clock uncertainty                            -0.45      12.85
u_int/int_flop_0/CLK (DFF)                    0.00      12.85 r
library setup time                           -0.12      12.73
data required time                                      12.73
```

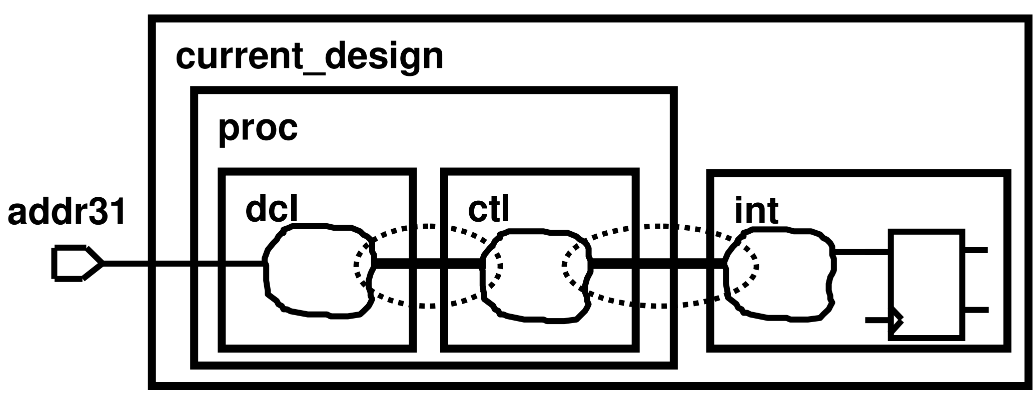
Identify two items which can potentially be addressed to improve the slack ….

slack (VIOLATED) -4.53

8- 12

1) Notice the large input delay of 8.4 ns relative to the clock period of 12.5ns – that's almost 70%! The input delay may be a conservative estimate which, in reality, is far smaller. Assuming your input comes from some other designer's block, you can discuss this spec with that designer to see if you can reduce your input delay value to a more reasonable number.

2) The sub-designs ***dcl***, ***ctl*** and ***int*** all contain combinational logic that is separated by hierarchy, which leads to sub-optimal synthesis. If possible, the partitioning of this design should be modified to remove the hierarchy along this combinational critical path.



Analysis Recommendations

After each compile or optimization step

- **Use `report_constraint -all` to determine <u>all</u> the constraint violations in your design**
- **Use `report_timing`, with appropriate options, to analyze violating timing paths in more detail**

8- 13

Summary: Commands Covered

```
report_timing
                   [ -delay max | min ]
                   [ -to pin_port_clock_list ]
                   [ -from pin_port_clock_list ]
                   [ -through pin_port_list ]
                   [ -group]
                   [ -input_pins ]
                   [ -max_paths path_count ]
                   [ -nworst paths_per_endpoint_count ]
                   [ -nets ]
                   [ -capacitance ]
                   [ -significant_digits number]
```

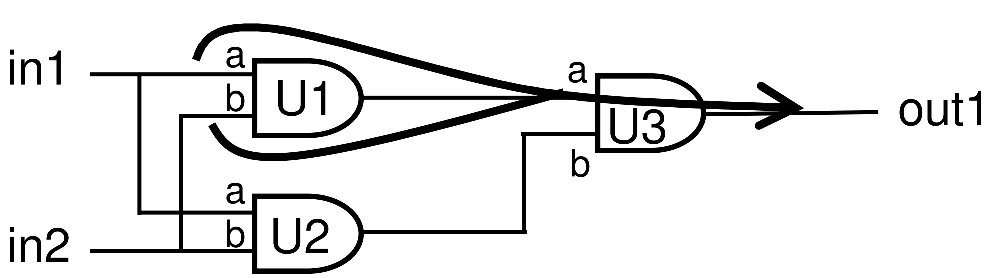
8- 14

Note:
Prior to v2007.03 only one `-through` option is allowed. Therefore, to specify multiple "though pins" these pins are included in a {list}. The following generates two reports – the user decides which is the worst path based on the results:

```
report_timing  -through {U1/a U3/a}
report_timing  -through {U1/b U3/a}
```

Starting with v2007.03 multiple `-through` options are allowed. The following generates one report for the worst of the two paths through `U1/a` **OR** `U1/b` **AND** `U3/a`:

```
report_timing -through {U1/a U1/b} -through {U3/a}
```



Summary: Unit Objectives

You should now be able to:

- **Generate timing reports, with additional command options as needed, to diagnose timing constraint violations**

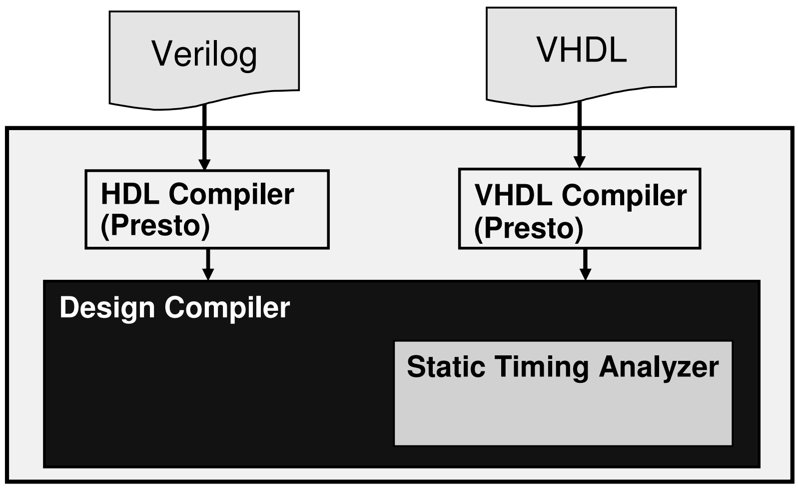
8- 15

Appendix

Static Timing Analysis Fundamentals

Static Timing Analysis: What Tool Do I Use?

- **Design Compiler has a built-in static timing analyzer**
- **STA is used**
  - During *compile* to guide optimization decisions
  - After *compile* to generate timing and timing-related reports



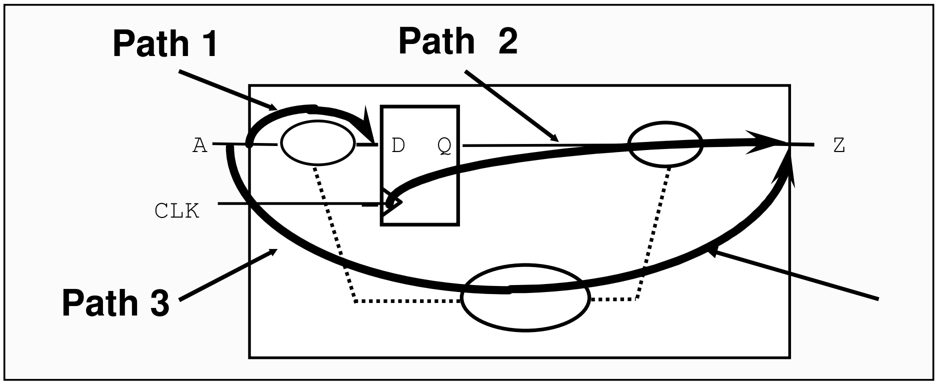
8- 17

PrimeTime is the Synopsys stand-alone, full-chip, gate-level, sign-off quality static timing analyzer:

- Runs faster than Design Compiler
- Can handle multi-million gate designs
- Uses the same libraries and netlist as Design Compiler
- Uses familiar Design Compiler commands as well as commands specific to PrimeTime, and is timing-consistent with Design Compiler
- Uses the Tcl tool command language
- Supports designs with multiple clocks and phases and gated clocks, multiple functional modes, multicycle paths, transparent latches, and time borrowing
- Supports minimum and maximum analysis, false path detection, mode analysis, case analysis, and time borrowing in level-sensitive, latch-based designs

See the *PrimeTime User Guide* for more details.

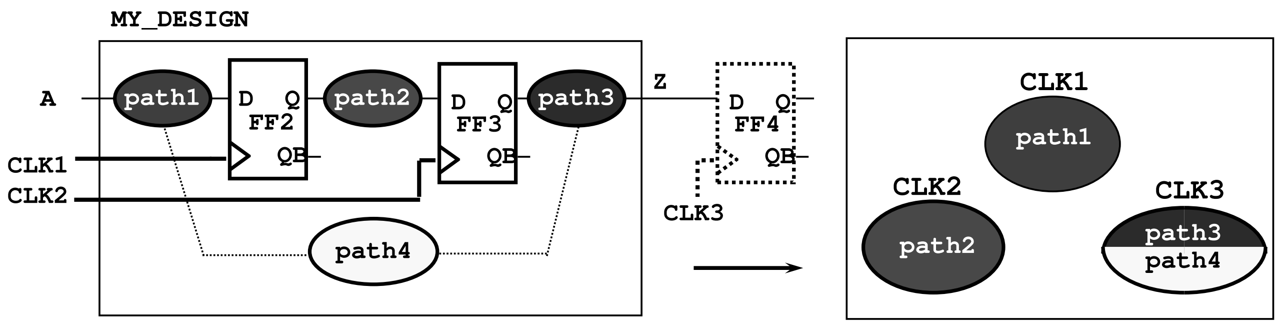
Static Timing Analysis



- **Static Timing Analysis can determine if a circuit meets timing constraints without dynamic simulation**
- **This involves three main steps:**
  - The design is broken down into timing paths
  - The delay of each path is calculated
  - All path delays are checked against timing constraints to determine if the constraints have been met

8- 18

Grouping of Timing Paths into Path Groups



Timing Paths **Path Groups**

Paths are grouped by the clocks controlling their <u>endpoints</u>

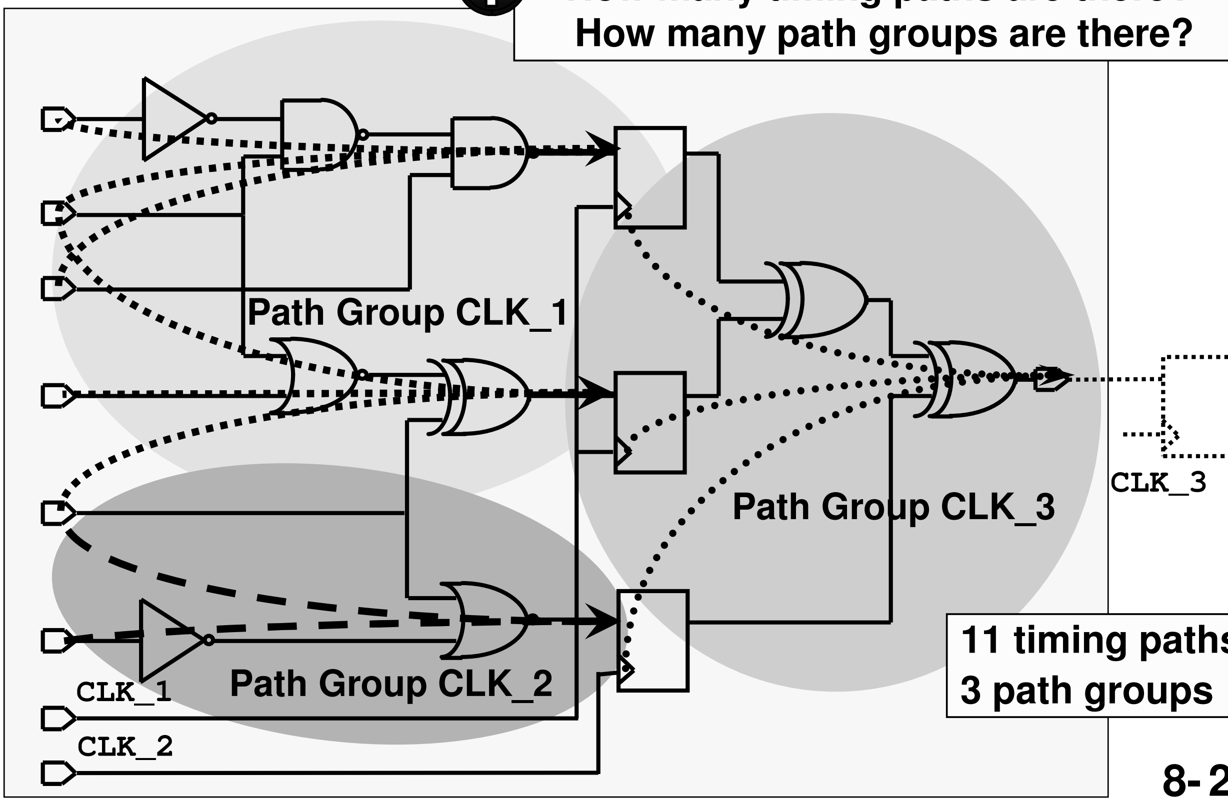
8- 19

`report_path_group:`

Reports information about the path groups in the current design.

Example: Timing Paths and Group Paths

How many timing paths are there?
How many path groups are there?



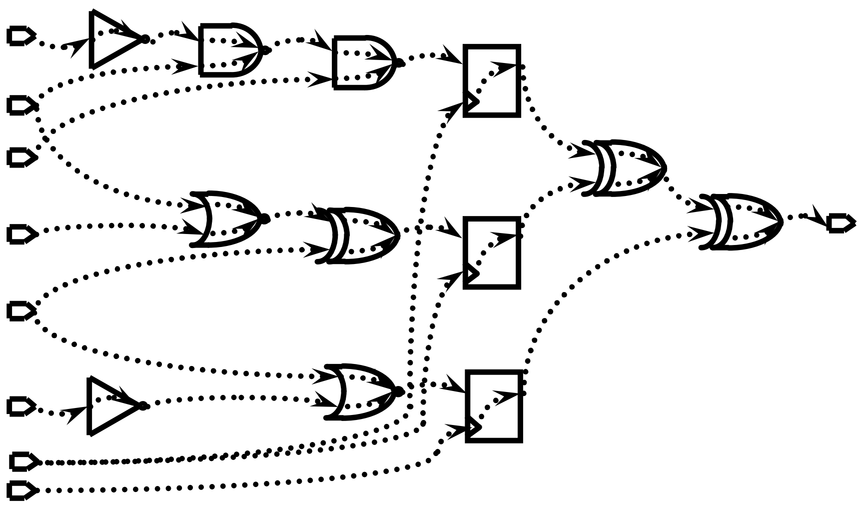
11 timing paths
3 path groups

8- 20

Schematic Converted to a Timing Graph

To calculate the total delay, Design Compiler breaks each path into timing cell and net delay arcs:

- Cell delays are typically calculated using non-linear delay models defined in the library
- Net delays are calculated based on WLMs or DC-Topographical estimates, and interconnect delay models defined in the library



8- 21

Non-linear delay models use Spice-derived timing at several *input_transition* and *output_load* points. Data-points not found in the tables are linearly interpolated. Note the presence of two tables: one for cell delay, and another for output transition (true rise/fall time). The output transition is needed in addition to the cell delay, because it is used as the input transition on the downstream cell to calculate its delay. Output transition is also used to check against the `max_transition` *design rule*, if applicable in the library. Input transition also affects output transition in this model, which affects the cell delay and output transition of the downstream cell, so a slow transition on one cell *ripple through* several downstream cells.

See the Library Compiler User Guide for more details.

Output load = 0.05 pF	Input transition = 0.5 ns
Cell Delay = .23 ns	Output Transition = 0.3 ns

SPICE	Output Load (pF)			
Input Trans (ns)	.005	.05	.10	.15
0.0	.10	.15	.20	.25
0.5	.15	.23	.30	.38
1.0	.25	.40	.55	.75

Cell Delay (ns)

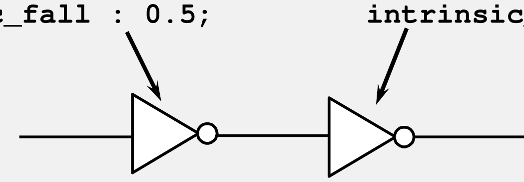
SPICE	Output Load (pF)			
Input Trans (ns)	.005	.05	.10	.15
0.0	.10	.20	.37	.60
0.5	.18	.30	.49	.80
1.0	.25	.40	.62	1.00

Output Transition (ns)

Edge Sensitivity in Path Delays

What is the longest path for the circuit below?

```
library: pin(Z)                       library: pin(Z)
 intrinsic_rise : 1.2;                 intrinsic_rise : 1.5;
 intrinsic_fall : 0.5;                 intrinsic_fall : 0.3;
```



- **There is an "edge sensitivity" (called *unateness*) in a cell's timing arc:**
  - Design Compiler keeps track of unateness in each timing path
  - At a minimum, timing analysis is performed twice for each path: Once for rising, and once for falling input data[1]

8- 22

Answer: 0.5 + 1.5 = 2.0 ns.

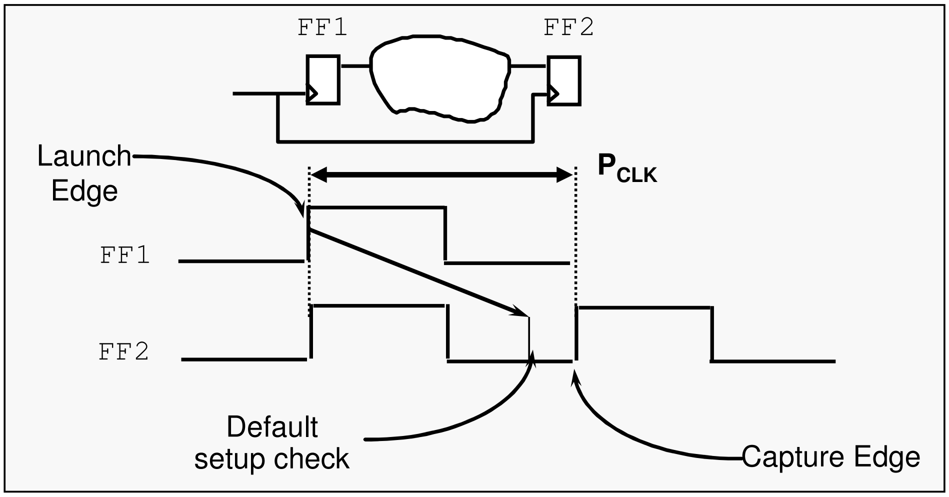
[1] If the path contains non-unate gates, e.g. XOR, XNOR, or the Select-to-output arc of a MUX, then the path is actually analyzed four times: The "other" input of the non-unate gate, the input that is NOT part of the timing path being analyzed, is held to logic 0 while the path is analyzed for rise- and fall-delay, and then repeated again while that "other" input is now set to logic 1.

Default Single-Cycle Behavior

By default Design Compiler assumes *single-cycle behavior*: all data is launched and captured within one clock cycle.

The path between FF1 and FF2 has a max delay constraint of:

$$\mathbf{P_{CLK} - FF2_{LibSetup}}$$



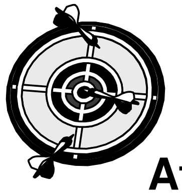
8- 23

This page was intentionally left blank.

Agenda

DAY 2

Unit Objectives

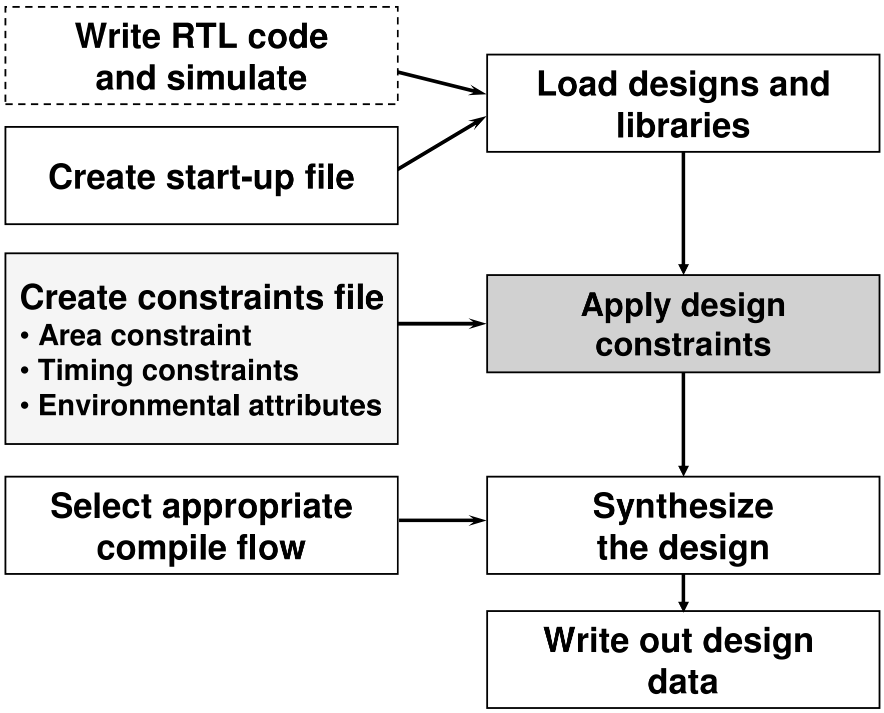


After completing this unit, you should be able to:

- **Apply setup timing constraints for designs with ‘non-default constraint’ conditions**

9- 2

RTL Synthesis Flow



9- 3

Constraints and Options To Be Covered

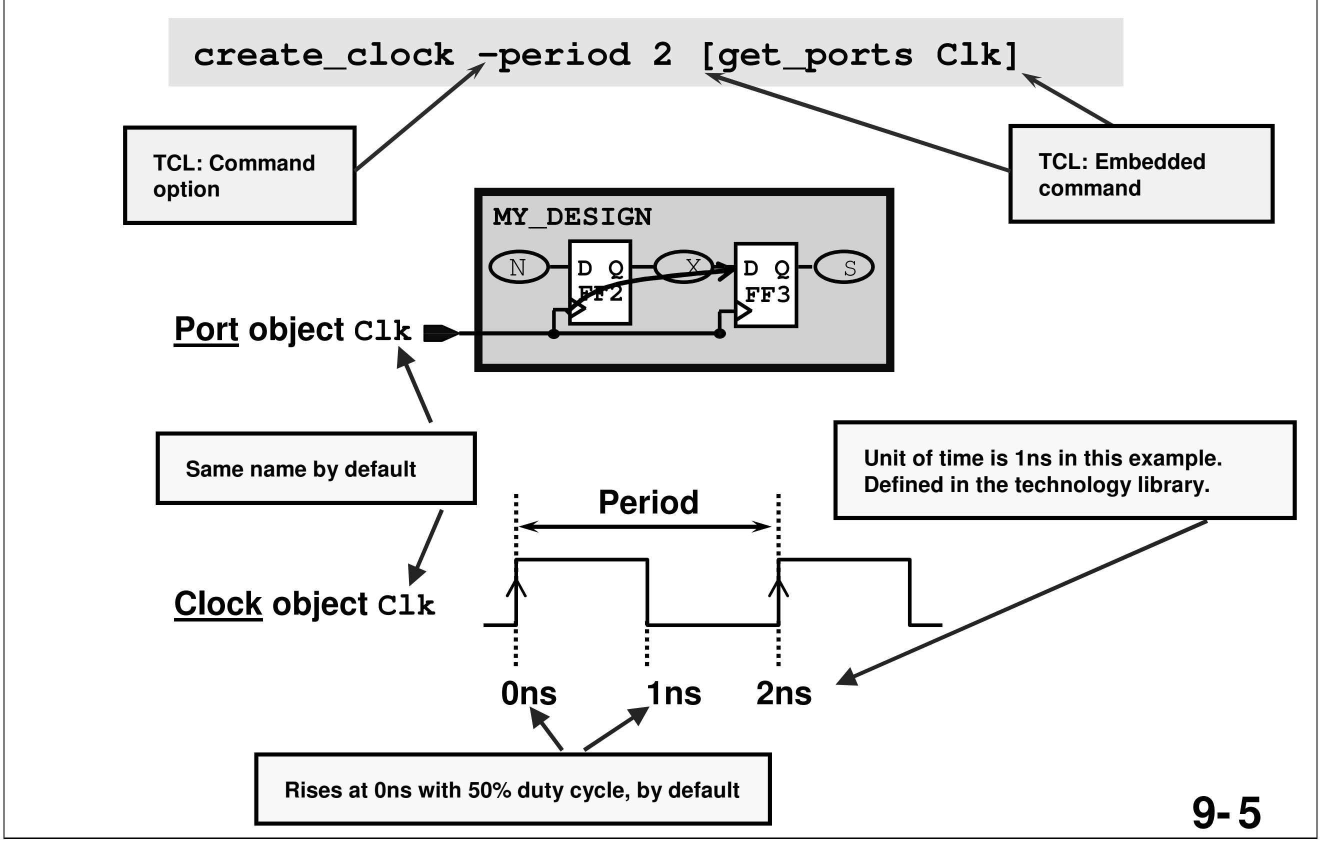
mydesign.con

```
create_clock -period 4 -name Ack -waveform {1 2} \
                                          [get_ports Clock1]
set_input_delay -max 0.48 -clock Clk [all_inputs]
set_input_delay -max 0.3 -clock Ack -network_latency_included \
                                    -clock_fall [get_ports A]
set_input_delay -max 1.2 -clock Ack -source_latency_included \
                                    -add_delay [get_ports A]
set_output_delay -max 0.8 -clock Ack [all_outputs]
set_output_delay -max 1.1 -clock Ack -add_delay -clock_fall \
                                          [get_ports OUT7]
set_load 0.080 [all_outputs]
set_load 0.035 [get_ports INPUT_C]
set_driving_cell -lib_cell FD1 -pin Q [all_inputs]
set_operating_conditions -max WCCOM
set_wire_load_model -name 1.6MGates
set_wire_load_mode enclosed
set_wire_load_model -name 200KGates [get_designs "SUB1 SUB2"]
set_wire_load_model -name 3.2MGates [get_ports IN_A]
set_port_fanout_number 8 [get_ports IN_A]
set_isolate_ports -type inverter [all_outputs]
```

9- 4

We will cover how `set_input_delay` together with `set_driving_cell` affect the data arrival time at input ports.

Defining a Clock: Recall Default Behavior

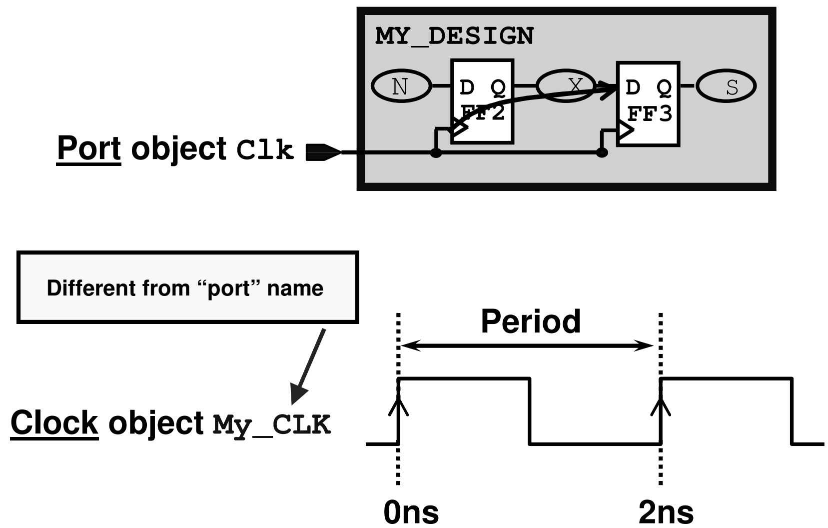


Unless explicitly defined with the `-waveform` option, the clock object that is created starts off going high at 0ns, comes down at 50% of the clock period, and repeats every clock period. The clock object's name is the same as the port/pin object to which it is assigned, unless explicitly defined by the `-name` option.

The unit of time is defined in the technology library. It is commonly 1 nano-second, as in the example above, but it may also be defined differently, for example: 0.1 nano-second, 10 pico-seconds, or 1 pico-second. You should verify the unit of time by looking at the top of the report generated by `report_lib <library_name>`.

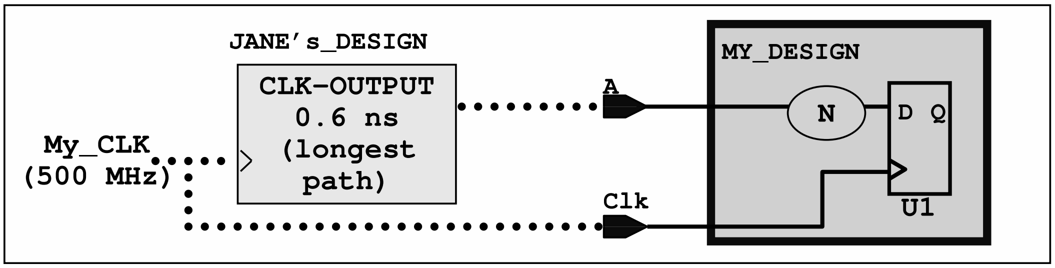
Defining a Clock: Different Clock Name

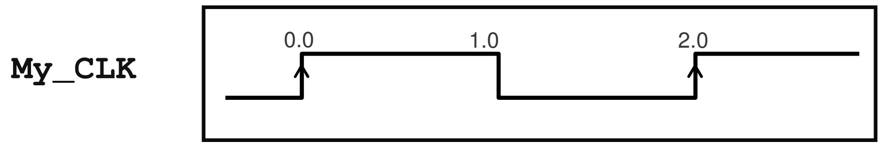
```
create_clock -period 2 -name My_CLK [get_ports Clk]
```



9- 6

Input Delay with Different Clock Name






```
create_clock -period 2 -name My_CLK [get_ports Clk]
set_input_delay -max 0.6 -clock My_CLK [get_ports A]
```

JANE's clock object name, not port name

9- 7

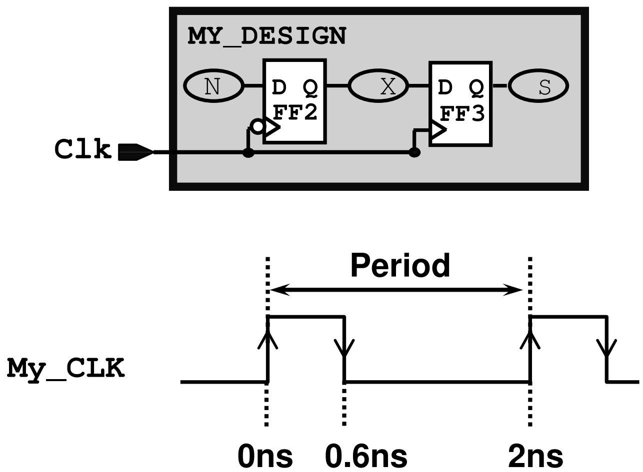
Defining a Clock: Duty-cycle

```
create_clock -period 2 -waveform {0 0.6} \
-name My_CLK [get_ports Clk]
```

TCL: Continued on next line

Rising edge

Falling edge

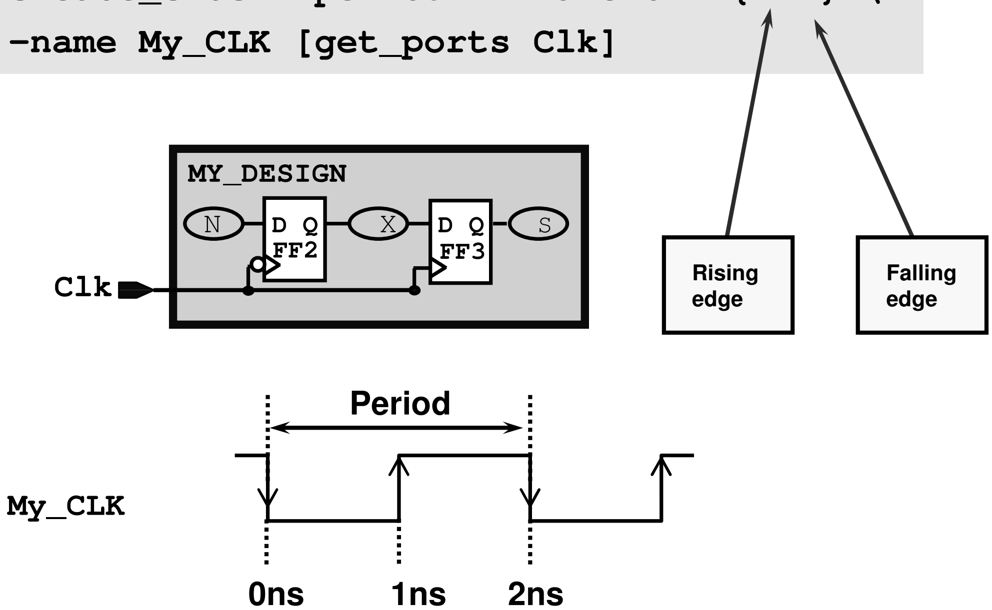


9- 8

This waveform has a 30% duty cycle.

Defining a Clock: Inverted

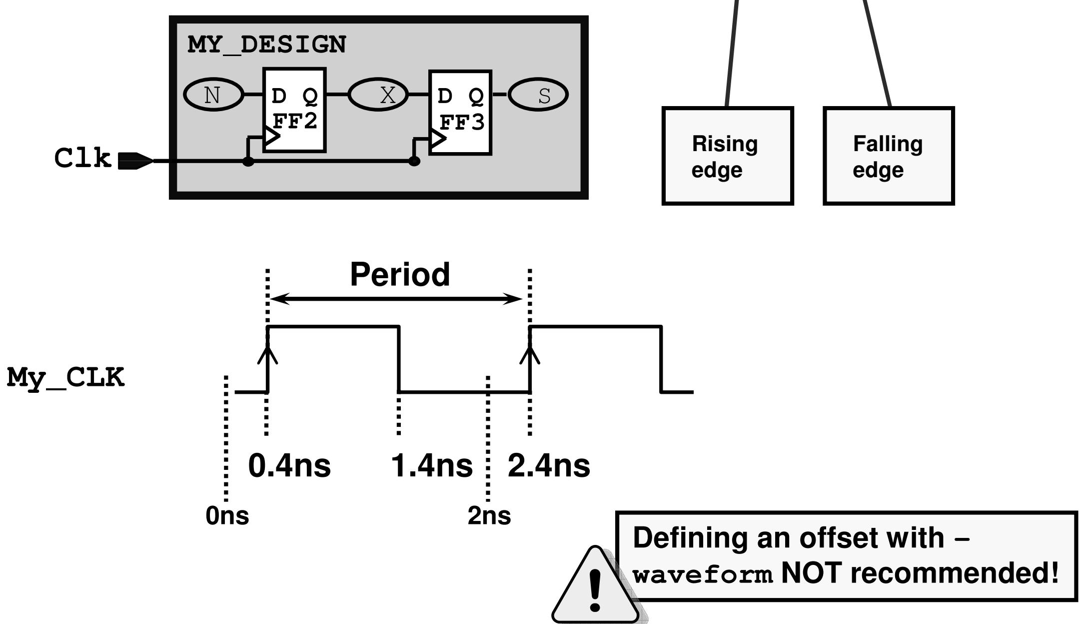
```
create_clock -period 2 -waveform {1 2} \
-name My_CLK [get_ports Clk]
```



9- 9

Defining a Clock: Offset

```
create_clock -period 2 -waveform {0.4 1.4} \
-name My_CLK [get_ports Clk]
```

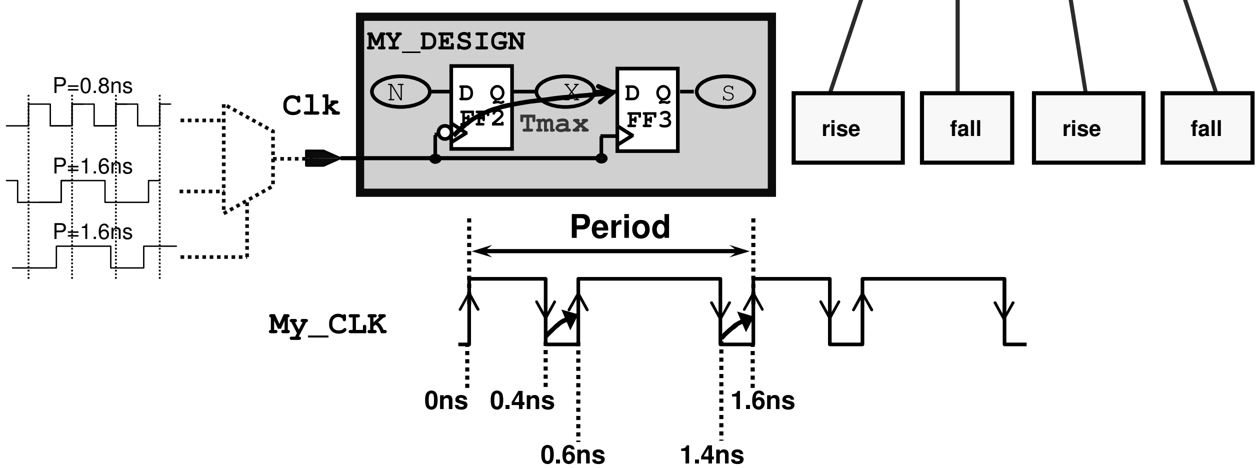


9- 10

It is NOT recommended to use the `-waveform` option to offset the clock waveform. In a multi-clock environment this offset method can cause DC to pick unintended clock launch and/or capture edges during synthesis which may over- or under-constrain your design. The `-waveform` option should be used only to modify the duty-cycle and/or invert the waveform. To introduce an offset it is recommended to use `set_clock_latency`.

Defining a Clock: Complex

```
create_clock -period 1.6 -waveform {0 0.4 0.6 1.4} \
-name My_CLK [get_ports Clk]
```



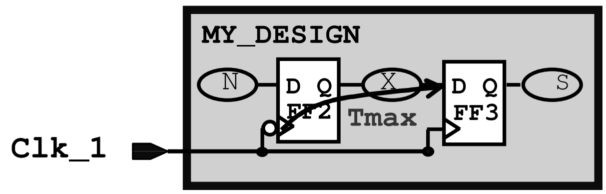
? **If $FF3_{setup}$ = 0.03ns, what is T_{max} for the Reg-to-Reg path?**

$T_{max,\ Reg\text{-}to\text{-}Reg}$ = ______________________________

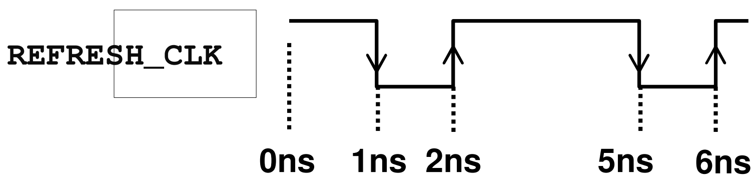
9- 11

Answer: 0.6 – 0.4 – 0.03 = 0.17ns.

Defining a Clock: Exercise



How do you define the following clock?



`create_clock` ______________________________

If FF3$_{setup}$= 0.3ns, what is T$_{max}$ for the Reg-to-Reg path?

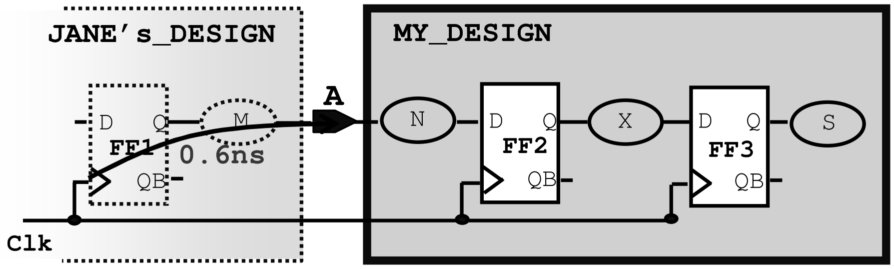
$T_{max,\ Reg\text{-}to\text{-}Reg}$ = ______________________

9- 12

```
create_clock -period 4 -waveform {2 5} \
-name REFRESH_CLK [get_ports Clk_1]
```

$T_{FF3,\ capture} - FF3_{Setup} - T_{FF2,\ launch} = 2 - 0.3 - 1 = 0.7$ ns.

Input Delay: Recall – Basic Options



External delay: JANE's maximum output delay

```
set_input_delay -max 0.6 -clock Clk [get_ports A]
```

Constrains MY input path for <u>setup</u> time

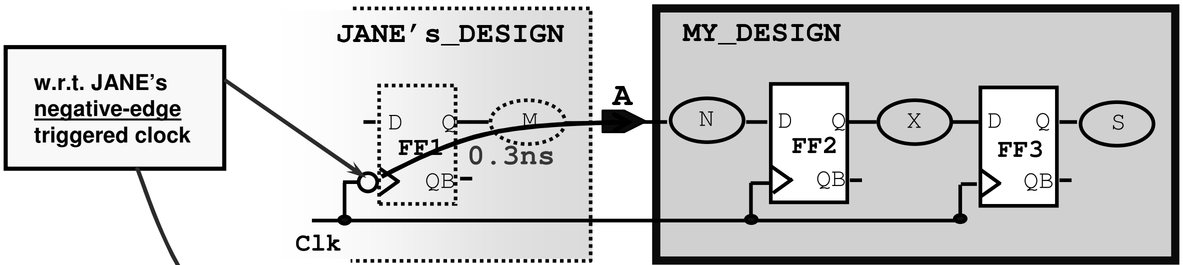
w.r.t. JANE's <u>positive-edge</u> triggered clock

You specify how much time is used by <u>external</u> logic...

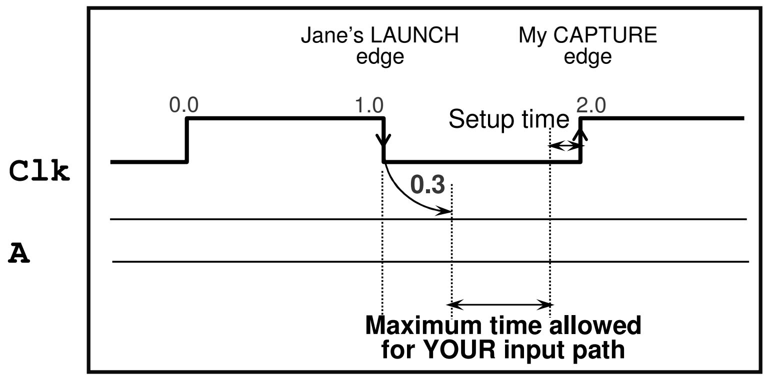
DC calculates how much time is left for the <u>internal</u> logic.

9- 13

Input Delay: Falling Clock Edge

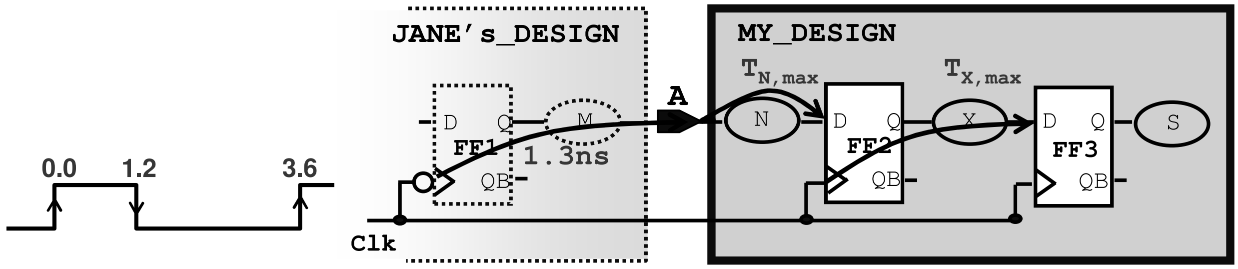



```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.3 -clock Clk -clock_fall [get_ports A]
```



9- 14

Input Delay: Falling Clock Edge Exercise




```
create_clock -period ____________________________[get_ports Clk]
set_input_delay __________________________________ [get_ports A]
```

?

If FF2 and FF3 have a 0.2 ns setup requirement:

What is the maximum delay $T_{N,\ max}$? ______________

What is the maximum delay $T_{X,\ max}$? ______________

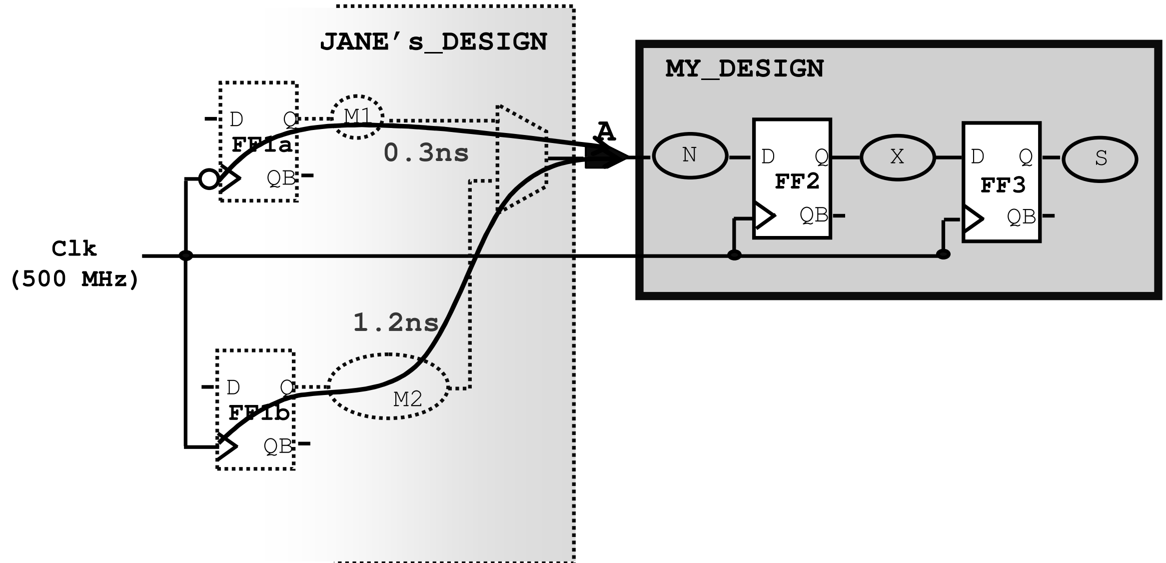
9- 15

```
create_clock -period 3.6 -waveform {0.0 1.2} [get_ports Clk]
set_input_delay -max 1.3 -clock Clk -clock_fall [get_ports A]
```

Tn, max = 3.6 – 1.2 – 1.3 - 0.2 = 0.9ns

Tx, max = 3.6 – 0.2 = 3.4ns

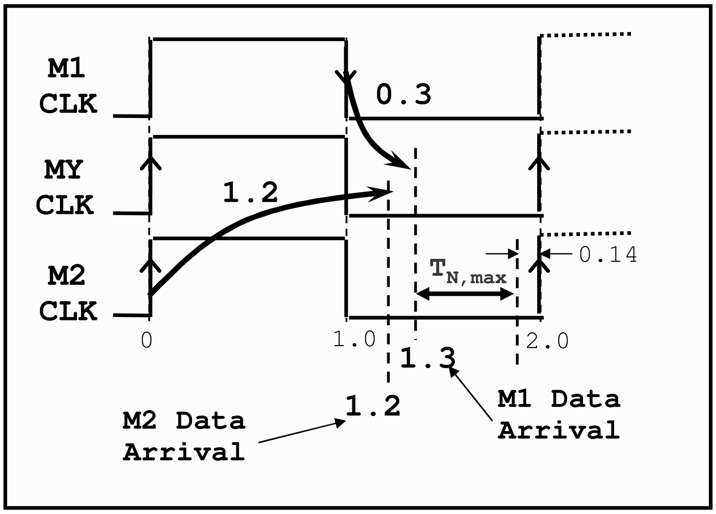
Input Delay: Multiple Input Paths




```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.3 -clock Clk -clock_fall \
[get_ports A]
set_input_delay -max 1.2 -clock Clk -add_delay [get_ports A]
```

9-16

Multiple Input Path Timing Analysis



Design Compiler analyzes both paths and constrains input logic path N with the more restrictive of the two – path M1 in this example

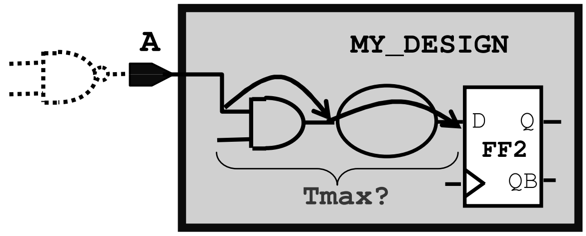
If FF2 has a 0.14 ns setup requirement:

What is the maximum delay $T_{N,\ max}$? ________________

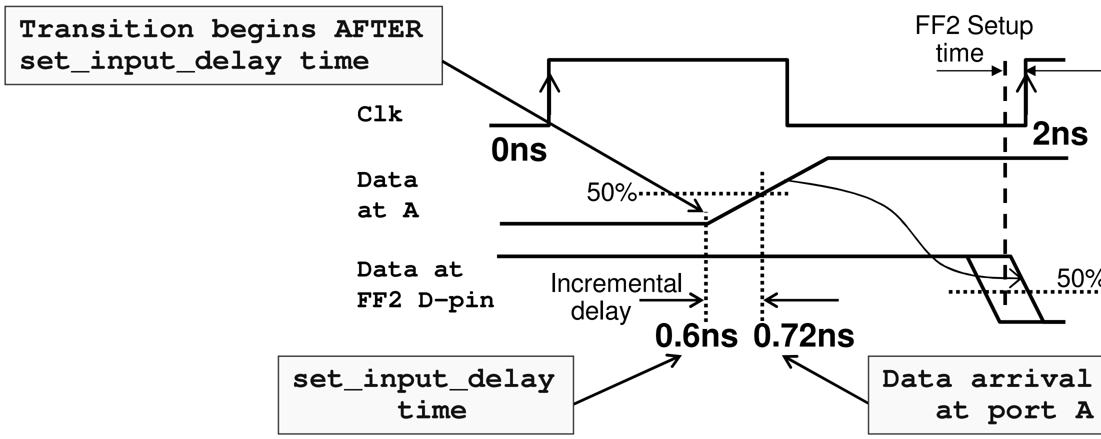
9- 17

Tn, max = 2.0 – 1.3 – 0.14 = 0.56ns

Effect of *Driving Cell* on Input Delay




```
set_input_delay -max 0.6 -clock Clk [get_ports A]...
set_driving_cell -lib_cell NAND2_3 [get_ports INPUT_A]...
```



9- 18

By specifying a driving cell DC calculates an accurate rise/fall transition time on the input port, based on your actual internal capacitive loading. This allows DC to calculate an accurate cell delay for the input gate, and the input path.

DC assumes, however, that the transition on the input port begins at the “set_input_delay time”. DC therefore adds an incremental delay to the data arrival time specified by `set_input_delay`.

set_driving_cell Recommendation

If your are confident that your design constraint specs are accurate, and you want to model your input data arrival time with precision, make sure that:

1. The `set_input_delay` number you apply is based on zero output load on Jane's block (the intrinsic delay to Jane's output port)
2. The `set_driving_cell` gate matches Jane's output driver

If not, your input constraints will include some built-in pessimism

Spec:

Latest Data Arrival Time at Port A, after Jane's launching clock:

0.60ns, with 50fF load

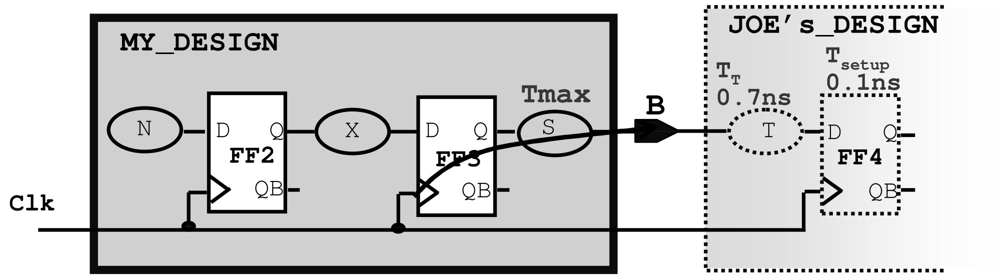
Use this number! → 0.48ns, with 0.0fF load

Driving cell on input port A: Qn pin of FD1 flip-flop

```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.48 -clock Clk [get_ports A]
set_driving_cell -lib_cell FD1 -pin Qn [get_ports A]
```

9- 19

Output Delay: Recall – Basic Options



External delay: JOE's maximum setup time: input delay (T) + FF setup time

```
set_output_delay -max 0.8 -clock Clk [get_ports B]
```

Constrains MY output path for maximum delay, or setup time w.r.t. Joe's input

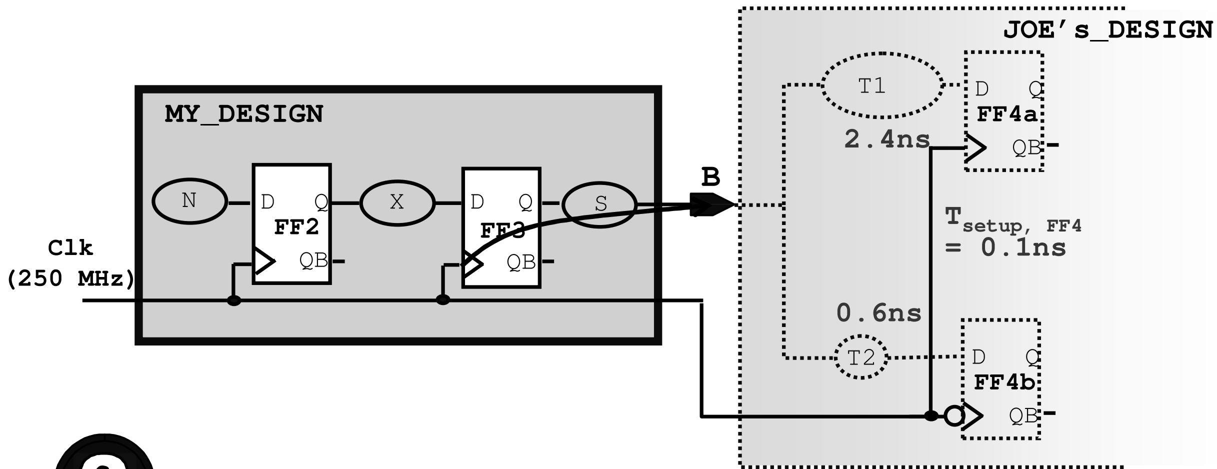
w.r.t. JOE's positive-edge triggered clock

Again, you specify how much time is used by external logic...

DC calculates how much time is left for the internal logic.

9- 20

Output Delay: Complex Output Paths



How do you constrain MY_DESIGN for the indicated path?

create_clock -period 4 [get_ports Clk]

set_output_delay ________________________________

set_output_delay ________________________________

9- 21

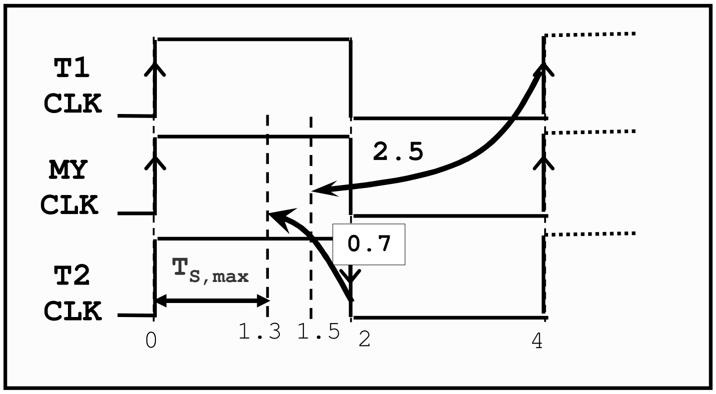
```
set_output_delay -max 2.5 -clock Clk [get_ports B]
set_output_delay -max 0.7 -clock Clk - clock_fall -add_delay \
                                              [get_ports B]
```

Or alternatively, if you recognize that the T2 path is the more restrictive one:

```
set_output_delay -max 0.7 -clock Clk -clock_fall [get_ports B]
```

Complex Output Path Timing Analysis

```
create_clock -period 4 [get_ports Clk]

set_output_delay -max 2.5 -clock Clk [get_ports B]

set_output_delay -max 0.7 -clock Clk -clock_fall \
-add_delay [get_ports B]
```



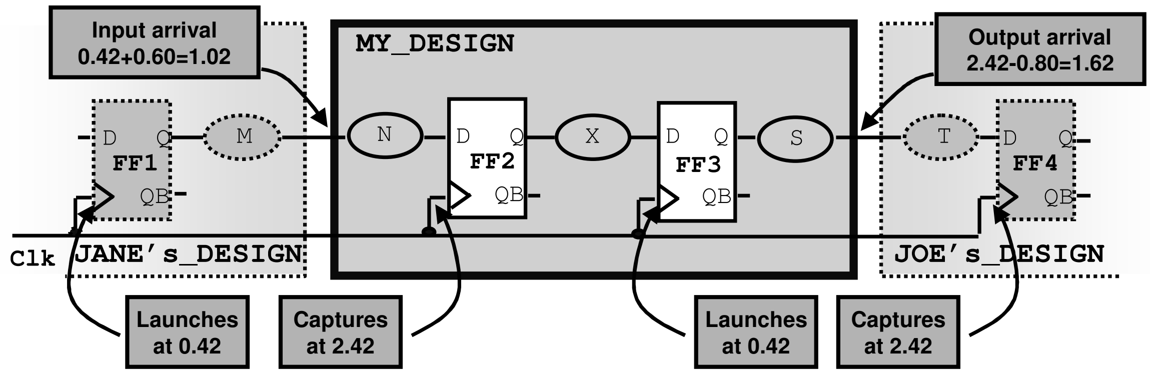
Design Compiler analyzes both paths and constrains output logic path S with the more restrictive of the two. $T_{S,\,max}$ = ____________

9- 22

$T_{s,\,max}$ = smaller of: (4.0 – 2.5) or (2.0 – 0.7) $\rightarrow$ = 1.3ns

Default External Clock Latencies

External registers, modeled by `set_in/output_delay` inherit the same network and source latencies as specified by the `set_clock_latency` commands, by default.

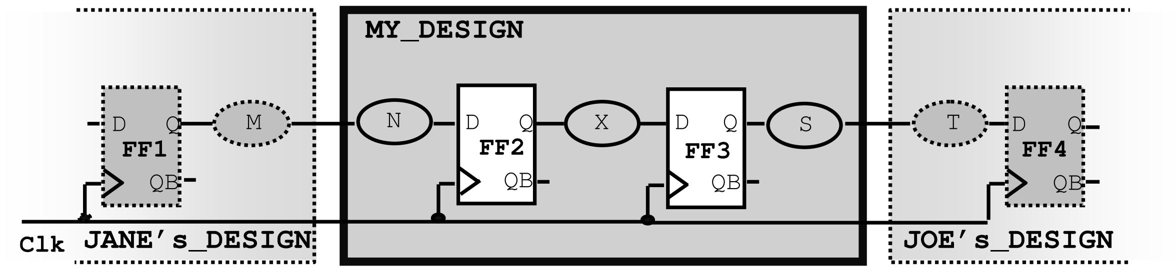



```
create_clock -period 2 [get_ports Clk]
set_clock_latency -source 0.3 [get_clocks Clk]
set_clock_latency 0.12 [get_clocks Clk]
set_input_delay -max 0.6 -clock Clk [all_inputs]
set_output_delay -max 0.8 -clock Clk [all_outputs]
```

9- 23

What if External Latencies are Different?

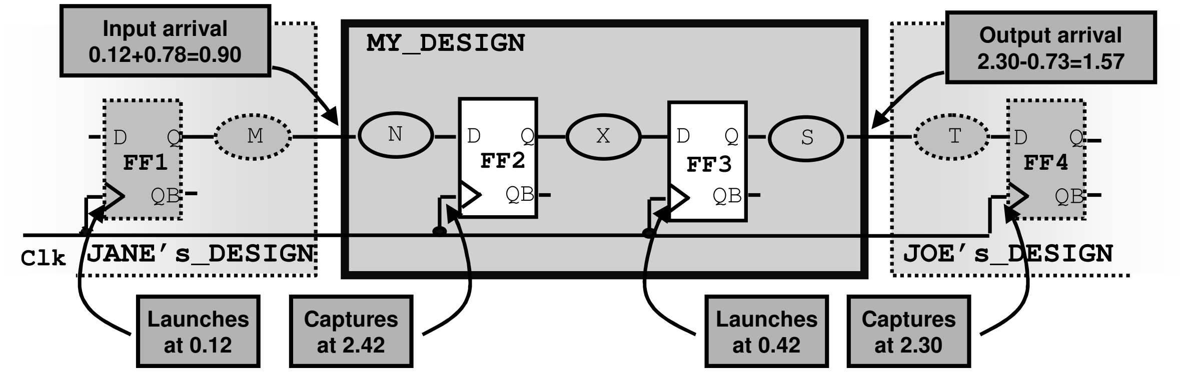
Spec:	JANE	ME	JOE
Source latency	0.18	0.30	0.30
Network latency	0.12	0.12	0.07



? How do we model non-default external latencies?

9- 24

“*Included*” External Clock Latencies



Spec:	**JANE**	**ME**	**JOE**
Source latency	0.18	0.30	0.30
Network latency	0.12	0.12	0.07

```
create_clock -period 2 [get_ports Clk]
set_clock_latency -source 0.3 [get_clocks Clk]
set_clock_latency 0.12 [get_clocks Clk]
set_input_delay -max [expr 0.6 + 0.18] -clock Clk \
                    -source_latency_included [all_inputs]
set_output_delay -max [expr 0.8 - 0.07] -clock Clk \
                    -network_latency_included [all_outputs]
```

9- 25

In the above example:

JANE’s_DESIGN has the same network latency as MY_DESIGN (0.12ns), but a different source latency of 0.18ns (vs. 0.30ns in MY_DESIGN). To model this:

1) We “keep” the `set_clock_latency 0.12` effect on MY_DESIGN’s input ports by doing nothing with it
2) We remove the incorrect source latency affect of `set_clock_latency -source 0.3` on MY_DESIGN’s input ports by using `set_input_delay … -source_latency_included`
3) We add the correct source latency to the input delay amount (`expr 0.6 + `**`0.18`**) to model the later arrival time of the input data due to the source latency

JOE’s_DESIGN has the same source latency as MY_DESIGN (0.30ns), but a different network latency of 0.07ns (vs. 0.12ns in MY_DESIGN). To model this:

1) We “keep” the `set_clock_latency -source 0.3` effect on MY_DESIGN’s output ports by doing nothing with it
2) We remove the incorrect network latency affect of `set_clock_latency 0.12` on MY_DESIGN’s output ports by using `set_output_delay … -network_latency_included`
3) We subtract the correct network latency from the output delay amount (`expr 0.8 `**`- 0.07`**) to model the later required arrival time of the output data.
(Remember: A larger output delay value means that the data must arrive more time before the capturing clock edge, or earlier; To model a later output data arrival time you must therefore decrease the output delay amount.

Argument Ordering of TCL Commands

```
set_input_delay -max 0.3 -clock Clk -clock_fall [get_ports A]
```

TCL:

Order of options does not matter. Source port or pin must be last.

SAME

```
set_input_delay -clock_fall -max -clock Clk 0.3 [get_ports A]
```

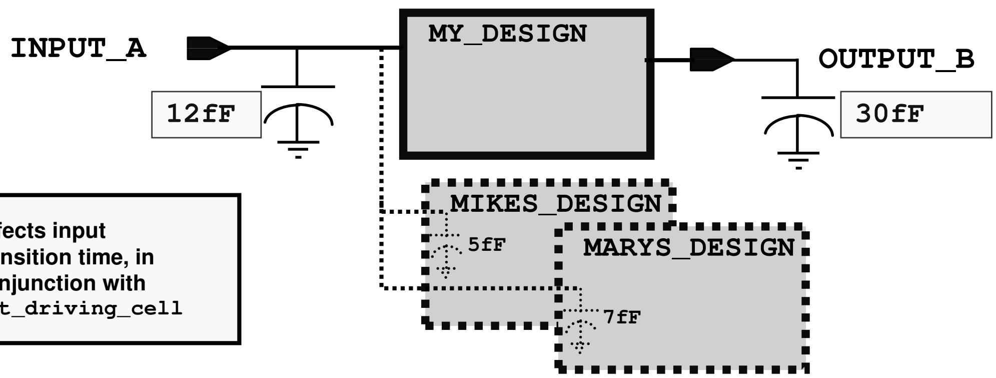
Port or pin must be the last argument

9- 26

Modeling External Capacitive Load on Inputs

Spec:

Block-level: Input A Drives two other block with 12fF total input capacitance



Affects input transition time, in conjunction with `set_driving_cell`

```
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk [get_ports A]
set_output_delay -max 0.8 -clock Clk [get_ports B]
set_load 0.030 [get_ports OUTPUT_B]
set_driving_cell -lib_cell NAND2_3 [get_ports INPUT_A]
set_load 0.012 [get_ports INPUT_A]
```

9- 27

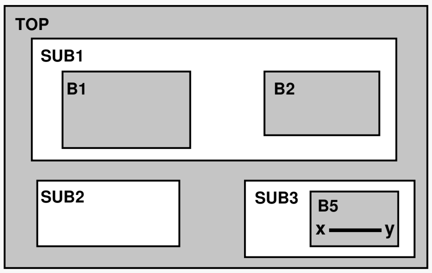
The additional load on the input port will affect the transition time on port INPUT_A.

If you have a `set_input_transition` on INPUT_A instead of `set_driving_cell`, then the `set_load` on the input port will have no effect on the input transition.

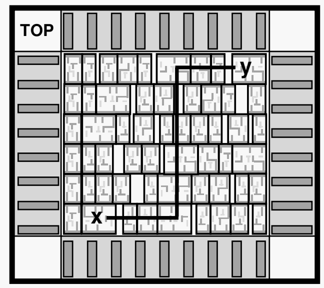
Recall: Wire Load Model

```
set_wire_load_model –name 1.6MGates
```

The specified model is applied to ALL nets at the current design level and below – OK if your physical layout is ‘flat’



Logical Design Hierarchy



‘Flat’ Physical Design

What if the physical layout is not flat?

9- 28

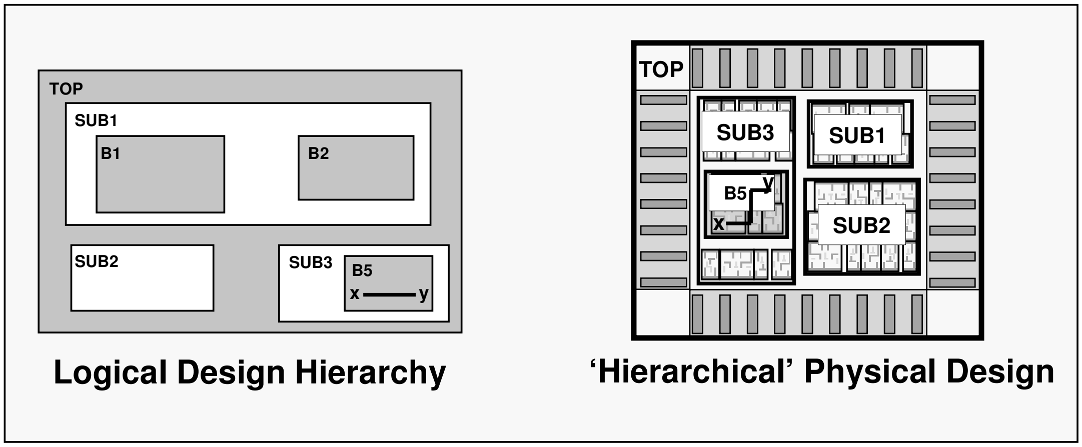
In the example above the net inside sub-block B5 which connects pin x to pin y seems like a relatively “small, local” net.

However, since the layout or standard cell placement of the physical design is ‘flat’, pin x and pin y can actually be placed physically far apart, and the metal interconnect can run across the full area of the entire chip.

Because of this possibility in a ‘flat’ layout you should use a conservative strategy and apply a “chip-level” or “top-design-level” wire load model to all the nets in the entire design.

Multiple WLMs in Hierarchical Designs

When the physical design is hierarchical – cells 'grouped' into physical 'sub-areas' – then smaller, individual wire load models can better model the RC characteristics of each sub-block



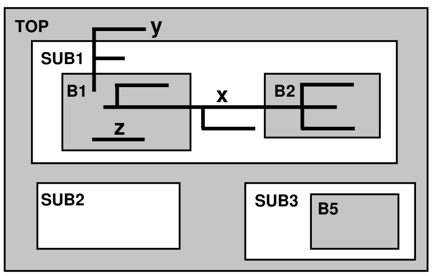
9- 29

In the example above the physical hierarchy closely matches that of the logical hierarchy. In this case it is not necessary to use the more conservative top-level wire load model for the entire design.

Specifying WLMs in a Hierarchical Design

- **Specify a WLM for each sub-block**
- **Set the *mode* to *enclosed***
  - Each net is assigned the smallest WLM that **encloses it**

```
current_design TOP
...
set_wire_load_model -name 1.6MGates
set_wire_load_mode enclosed
set_wire_load_model -name 800KGates [get_designs SUB1]
set_wire_load_model -name 200KGates [get_designs B1]
set_wire_load_model -name 100KGates [get_designs B2]
...
```



What WLM is selected for net:

x ____________________

y ____________________

z ____________________

9- 30

Mode **top** (the default mode unless otherwise specified) ignores lower level wire load models. Use this mode if your design will be layed out 'flat'.

Mode **enclosed** uses the lowest-level WLM that completely encloses the net. It is less pessimistic than the *top* mode and more realistic for a design that will be layed out 'hierarchically'.
In the example above the following WLMs are selected:
  x: 800KGates
  y: 1.6MGates
  z: 200KGates

Note: You need to understand what your physical layout (or floorplan) will look like to determine the best *mode* to use.

Mode **segmented** is not recommended because it tends to give optimistic results. It selects a WLM for each segment of a net that fans out in a different WLM area. E.g., for net **x** above DC will calculate an RC for: the B1 segment using 200KGates and a fanout of 1; the SUB1 segment using 800KGates and a fanout of 1; and the B2 segment using 100KGates and a fanout of 3.

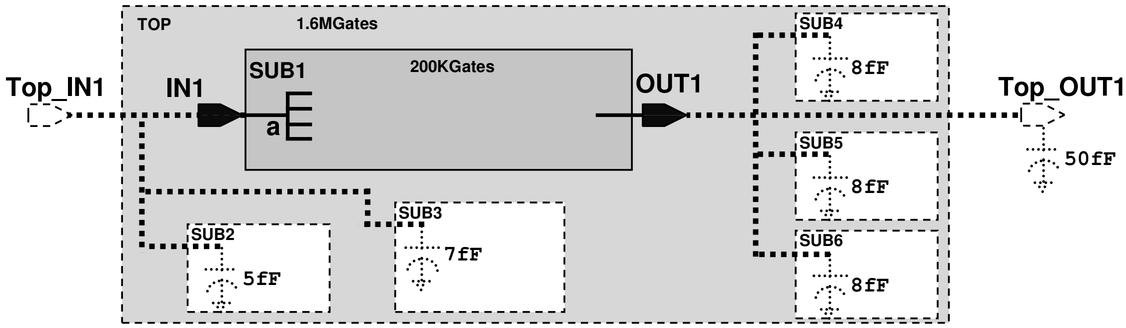
Use `report_wire_load` to display the wire load models that are applied to you design and its sub-blocks.

Your technology library may have an explicit default wire load model mode defined. To display its value use the following command. If no default attribute is specified, the default mode is *top*:

```
get_attribute <lib_name> default_wire_load_mode
```

Specifying Different PORT Wire Load Models

```
current_design SUB1
...
set_wire_load_model -name 200KGates
set_wire_load_model -name 1.6MGates [get_ports {IN1 OUT1}]
set_port_fanout_number 2 [get_ports IN1]
set_port_fanout_number 4 [get_ports OUT1]
set_load 0.012 [get_ports IN1]
set_load 0.074 [get_ports OUT1]
...
```



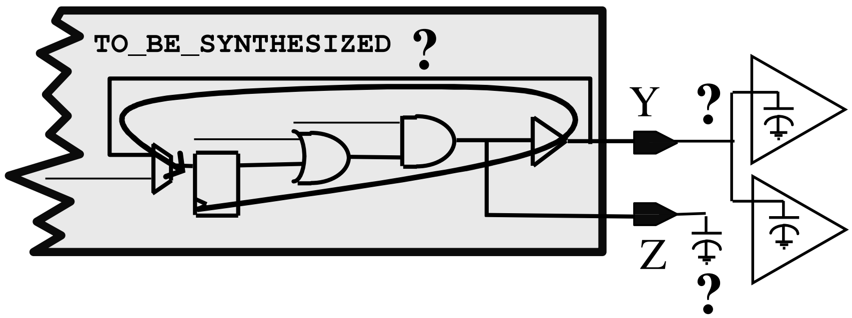
Ports *IN1* and *OUT1* are assigned the *TOP* design's WLM and the external fanouts are specified to calculate more accurate wire lengths.

9- 31

In the example above the sub-design *SUB1* is being constrained to be compiled. It is assumed that the layout will be hierarchical, so an appropriate WLM, *200KGates*, is selected for the block. However, it is also known that ports *IN1* and *OUT1* will actually be connected to top-level ports, which also fan out to other sub-blocks inside of *TOP*. It would be too optimistic to apply *SUB1's* WLM to the *IN1* and *OUT1* nets. It is more realistic to use the *TOP* design's WLM, *1.6MGates*, for these nets, while also taking into account the additional external fanout of the ports. Remember that the wire length in a WLM is a function of the fanout of the net.

Problem: Internal Paths with External Loads

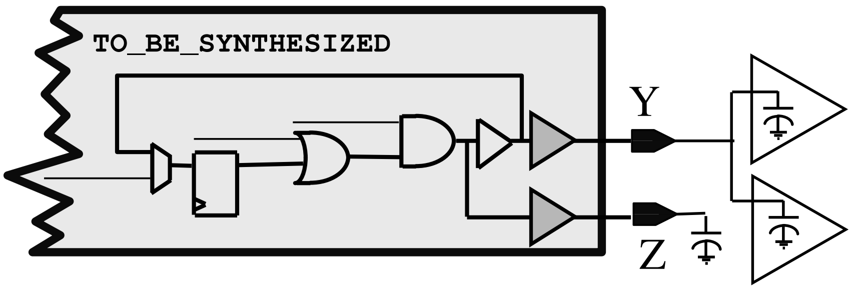
The problem:
The delay of the internal reg-to-reg path is a function of the external loads, which may not be known or accurate.



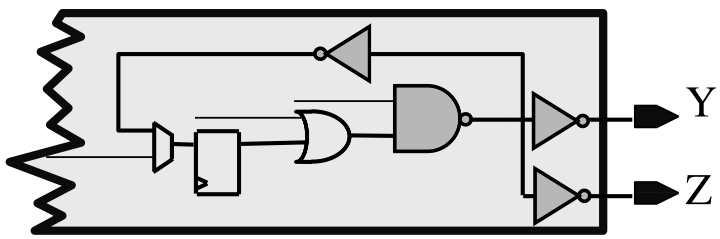
9- 32

Solution: Isolate Ports from External Loads

```
set_isolate_ports [all_outputs]
compile -b -scan -map high
```




```
set_isolate_ports -type inverter [all_outputs]
compile -b -scan -map high
```



9- 33

For an output port the compile directive inserts an isolation cell, if needed, to ensure that the net driving the port does not have any other internal fanout. For an input port an isolation cell is inserted if the port is driving multiple cells or if the port is driving a pin of a cell that contains more than one input pin. By default, no action is taken if the above conditions are not met. If the `-force` option is specified, isolation is performed on the port even if these conditions are not met.

By default `set_isolate_ports` uses buffers to isolate the ports. During optimization DC will optimize the output logic to try to meet timing constraints, but this may be limited by the use of buffers only. If `-type inverters` is specified, DC uses inverter pairs to isolate the ports, which can sometimes improve timing, as shown above (.

`report_isolate_ports:` Reports where buffers were inserted.

Summary: Constraints and Options Covered

mydesign.con

```
create_clock -period 4 -name Ack -waveform {1 2} \
                                          [get_ports Clock1]
set_input_delay -max 0.48 -clock Clk [all_inputs]
set_input_delay -max 0.3 -clock Ack -network_latency_included \
                                    -clock_fall [get_ports A]
set_input_delay -max 1.2 -clock Ack -source_latency_included \
                                    -add_delay [get_ports A]
set_output_delay -max 0.8 -clock Ack [all_outputs]
set_output_delay -max 1.1 -clock Ack -add_delay -clock_fall \
                                          [get_ports OUT7]
set_load 0.080 [all_outputs]
set_load 0.035 [get_ports INPUT_C]
set_driving_cell -lib_cell FD1 -pin Q [all_inputs]
set_operating_conditions -max WCCOM
set_wire_load_model -name 1.6MGates
set_wire_load_mode enclosed
set_wire_load_model -name 200KGates [get_designs "SUB1 SUB2"]
set_wire_load_model -name 3.2MGates [get_ports IN_A]
set_port_fanout_number 8 [get_ports IN_A]
set_isolate_ports -type inverter [all_outputs]
```

9- 34

Unit Objectives Review

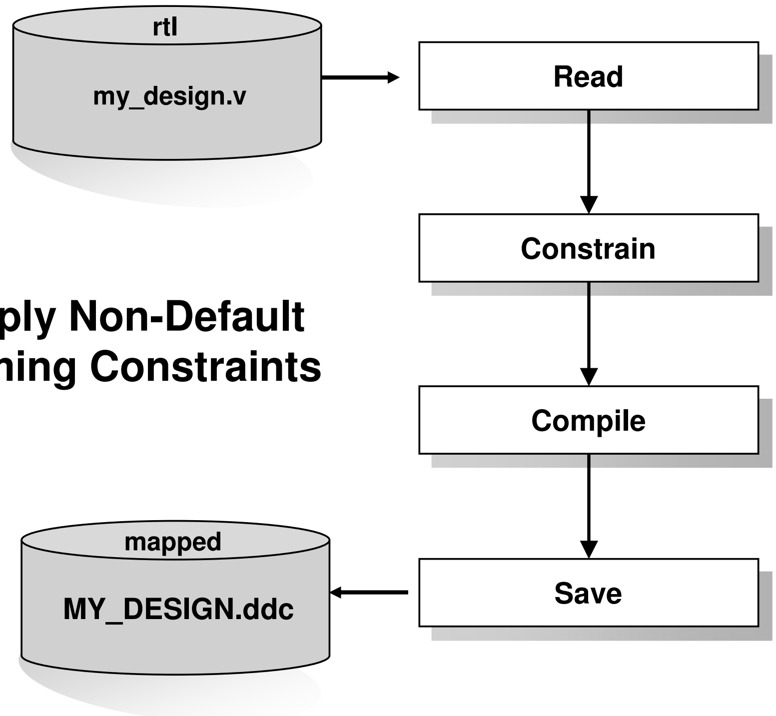
You should now be able to:

- **Apply setup timing constraints for designs with ‘non-default constraint’ conditions**

9- 35

Lab 9: More Constraint Considerations

90 minutes



Apply Non-Default Timing Constraints

9- 36

Agenda

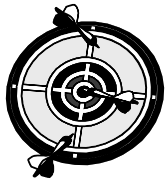
DAY 3

9	More Constraint Considerations (Lab cont'd)
10	Multiple Clock/Cycle Designs
11	Synthesis Techniques and Flows
12	Post-Synthesis Output Data
13	Conclusion

Synopsys 10-I-011-SSG-013

10- 1

Unit Objectives

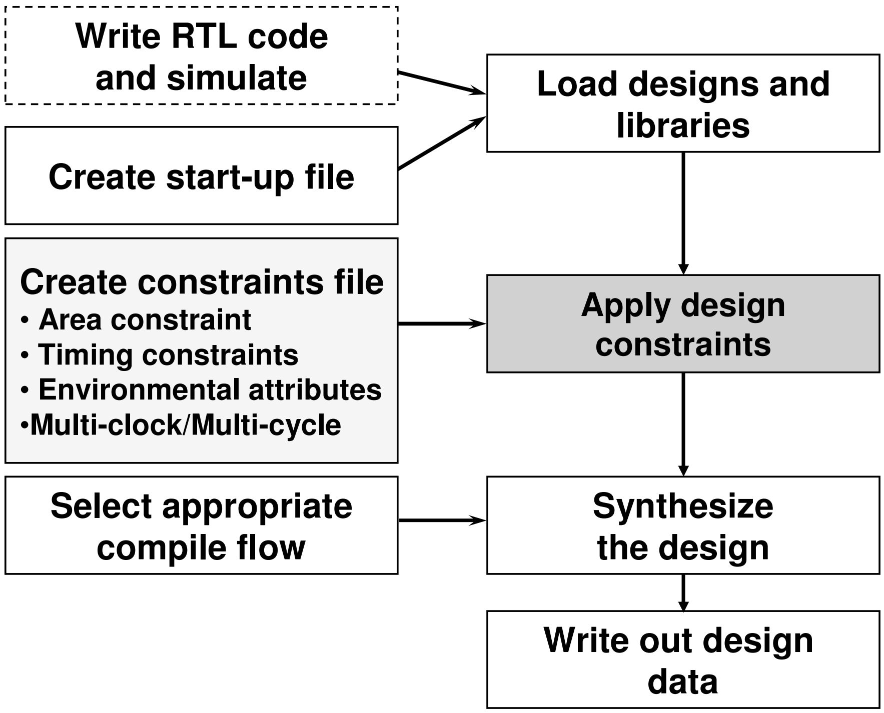


After completing this unit, you should be able to:

- **Constrain a synchronous multi-clock design**
- **Constrain an asynchronous multi-clock design**
- **Constrain a multi-cycle design**

10- 2

RTL Synthesis Flow



10- 3

Commands To Be Covered

mydesign.con

```
set_false_path -from [get_clocks CLKA] -to [get_clocks CLKB]

set_multicycle_path -setup 2 -from -from A_reg \
                    -through U_Mult/Out -to B_reg
set_multicycle_path -hold 1 -from -from A_reg \
                    -through U_Mult/Out -to B_reg

report_timing_requirements
report_timing_requirements -ignored
reset_path -from FF1/Q
```

10- 4

We will also cover how `set_input_delay` together with `set_driving_cell` affect the data arrival time at input ports.

Multiple Clocks: Synchronous

Multiple Clocks - Synchronous

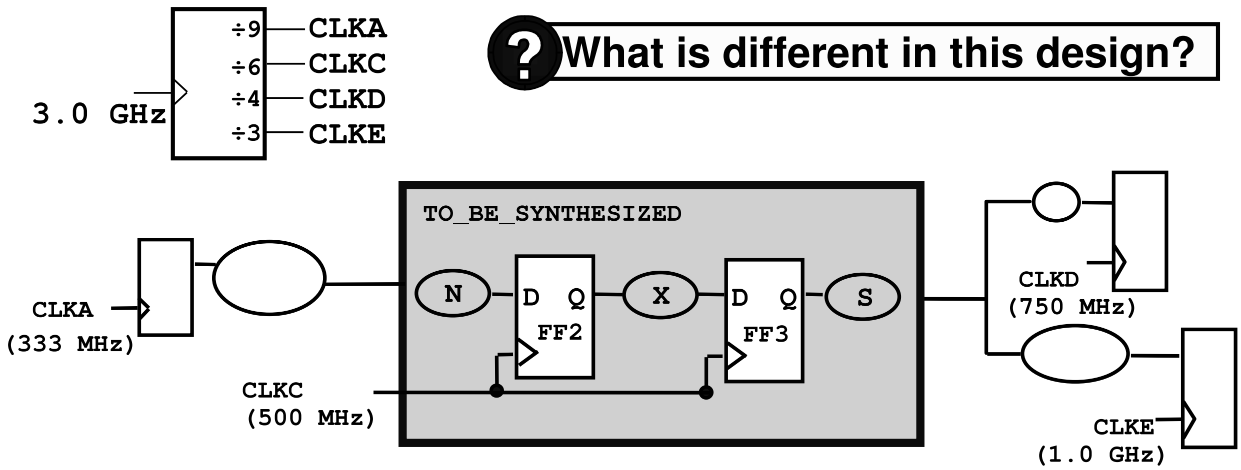
Multiple Clocks - Asynchronous

Multi-Cycle Path Constraints

Multi-Path Constraints

10- 5

Synchronous Multiple Clock Designs

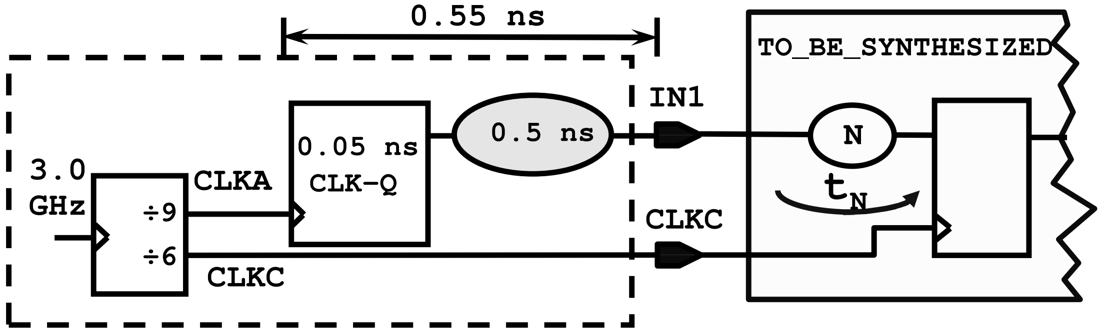


- Multiple clock sources
- All derived from the same clock source
- Some clocks do not have a corresponding clock port on our design
- Multiple constraints on a single port

10- 6

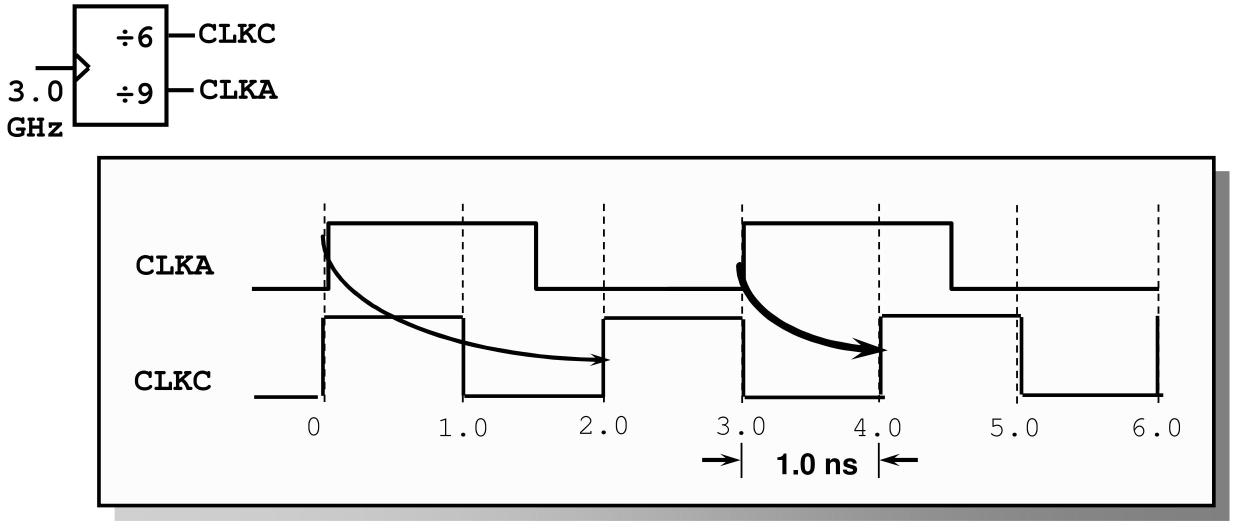
Multiple Clock Input Delay

```
create_clock -period 3.0 -name CLKA
create_clock -period 2.0 [get_ports CLKC]

set_input_delay 0.55 -clock CLKA -max [get_ports IN1]
```



10-7

Maximum Internal Input Delay Calculation



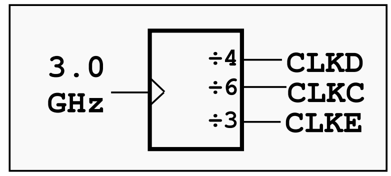
For the example shown, input logic cloud *N* must meet:

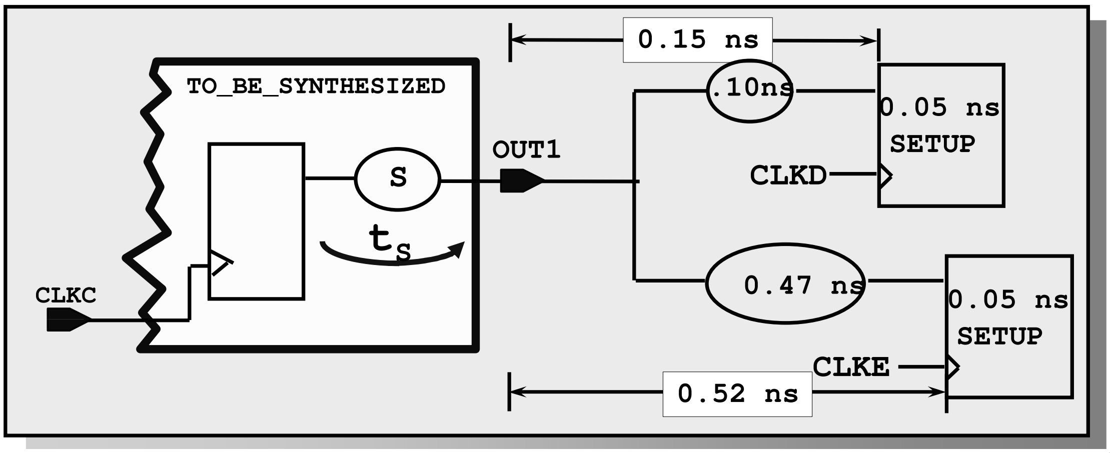
$$t_N < 1.0 - 0.55 - t_{setup}$$

10- 8

Multiple Clock Output Delay: Example

```
create_clock -period [expr 1000/750.0] -name CLKD
create_clock -period 1.0    -name CLKE
create_clock -period 2.0 [get_ports CLKC]

set_output_delay -max .15 -clock CLKD [get_ports OUT1]
set_output_delay -max .52 -clock CLKE -add_delay [get_ports OUT1]
```



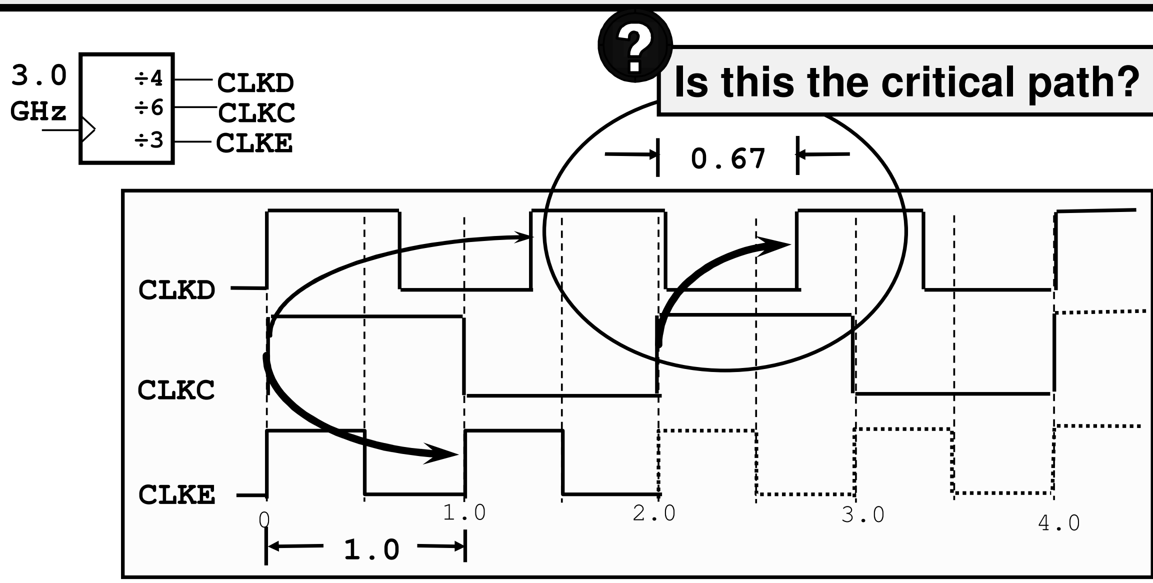


10- 9

The frequency for clock CLKD is 750MHz (3 GHz/4). For calculating the clock period, make sure at least one of the operands is a real number so that the answer is also a real number for the clock period. The result of [expr 1000/75] would be 13, instead of 13.333.

Without the `-add_delay` option, the second `set_output_delay` would over-write the first one. This is not what we want. We want DC to consider both paths and to constrain the output logic path S to meet both end-point constraints.

Maximum Internal Output Delay Calculation



The path through output cloud *S* must meet the smaller of:

$t_s < 1.0 - 0.52$	**AND**	$t_s < 0.67 - 0.15$
0.48 ns		0.52 ns

10-10

Summary: Multiple Clock Design

- **By definition, all clocks used with Design Compiler are synchronous**
  - You cannot create asynchronous clocks with the `create_clock` command
- **Constrain your design as usual – DC does the rest**
  - Use *virtual clocks* and `-add_delay` as needed
- **DC builds a common base period for all clocks related to each path**
- **DC then determines every possible data launch and capture time, and optimizes each path to the most conservative constraint**

10- 11

The base period is the least common multiple of all clock periods.

Multiple Clocks: Asynchronous

Multiple Clocks - Synchronous

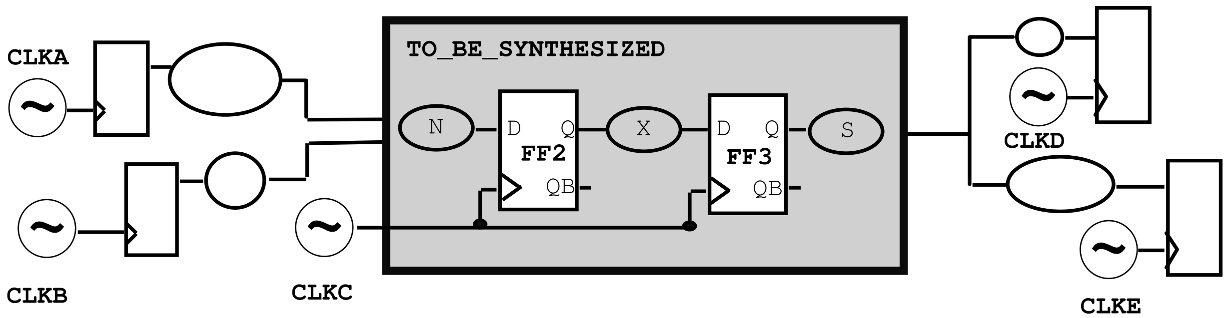
Multiple Clocks - Asynchronous

Multi-Cycle Path Constraints

Multi-Path Constraints

10- 12

Asynchronous Multiple Clock Designs



What do you do if the design has *asynchronous* clock sources?

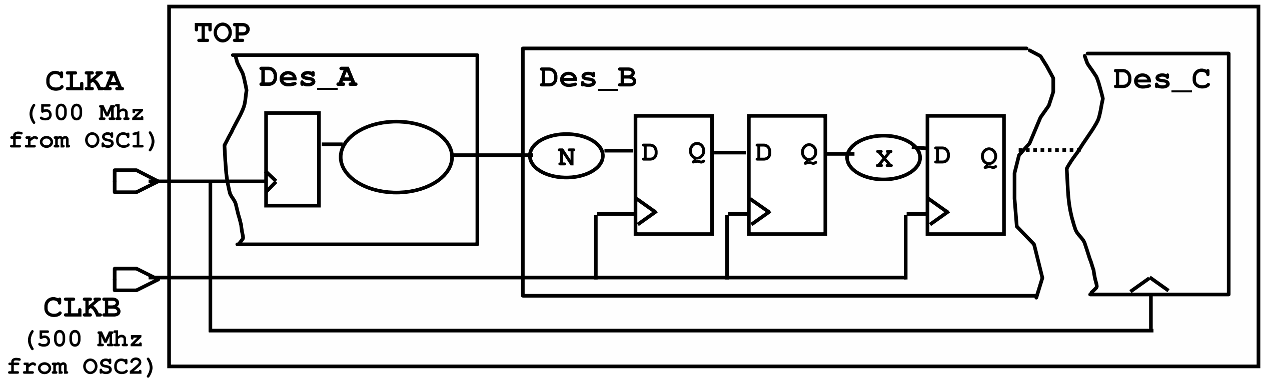
10- 13

Synthesizing with Asynchronous Clocks

- **It is your responsibility to account for the metastability:**
  - Instantiate double-clocking, metastable-hard Flip-Flops
  - dual-port FIFO, etc
- **Create clocks to constrain the paths within each clock domain**
- **You must also disable timing-based synthesis on any path which crosses an asynchronous clock boundary:**
  - This will prevent DC from wasting time trying to get the asynchronous path to "meet timing"

10- 14

Example: Asynchronous Design Constraints




```
current_design TOP
# Make sure register-register paths meet timing
create_clock -period 2  [get_ports CLKA]
create_clock -period 2  [get_ports CLKB]
...
# Don't optimize logic crossing clock domains
set_false_path -from [get_clocks CLKA] -to [get_clocks CLKB]
set_false_path -from [get_clocks CLKB] -to [get_clocks CLKA]
...
compile_ultra -scan -timing
```

10- 15

If all clocks in the design are asynchronous, use the following example.

```
set DESIGN_CLOCKS [all_clocks]
foreach_in_collection _CLK $DESIGN_CLOCKS {
  set_false_path -from $_CLK -to [remove_from_collection [all_clocks] $_CLK]
}
```

Constraining Multi-Cycle Paths

Multiple Clocks - Synchronous

Multiple Clocks - Asynchronous

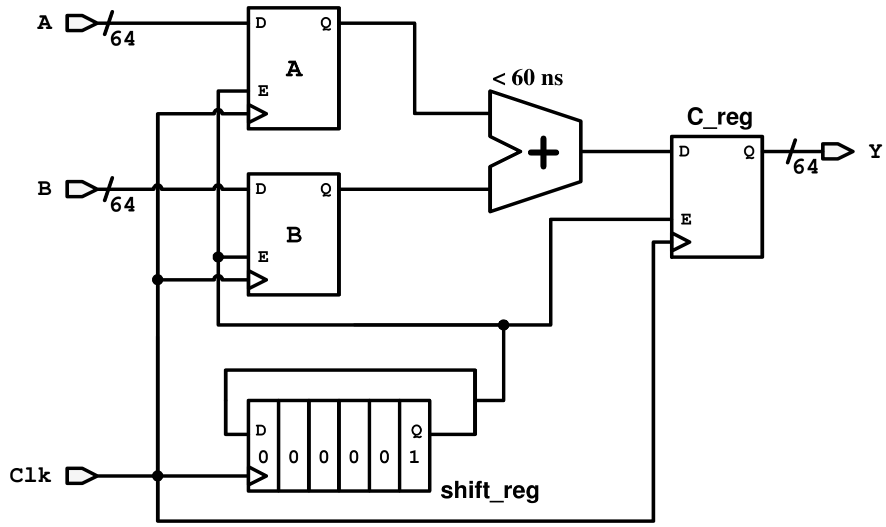
Multi-Cycle Path Constraints

Multi-Path Constraints

10- 16

Example Multi-cycle Design

Clock period is 10 ns. The adder takes almost 6 clock cycles. What happens when you apply `create_clock –period 10 Clk`?



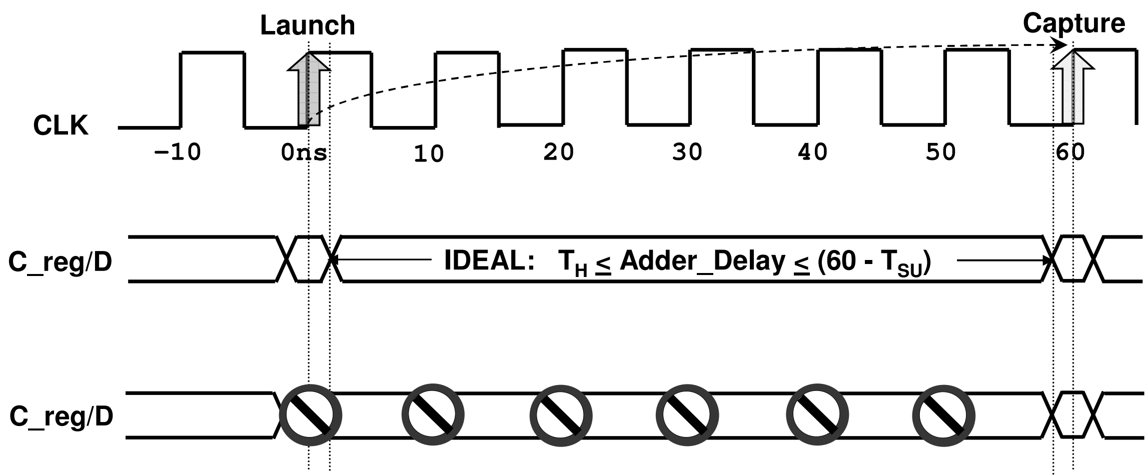
What's the solution?

10- 17

By specifying a `create_clock` with a period of 10ns the non-mullti-cycle paths are constrained correctly. However, since DC does not perform logic analysis it does not consider the fact that the registers above are enabled every sixth cycle only. DC therefore assumes that the paths through the adder must also meet the 10ns constraint!

Timing with Multi-cycle Constraints

```
create_clock –period 10 [get_ports CLK]
set_multicycle_path –setup 6 –to C_reg[*]
```



DC assumes change could occur near *any* clock edge causing metastability!

Where does DC perform hold analysis?

10- 18

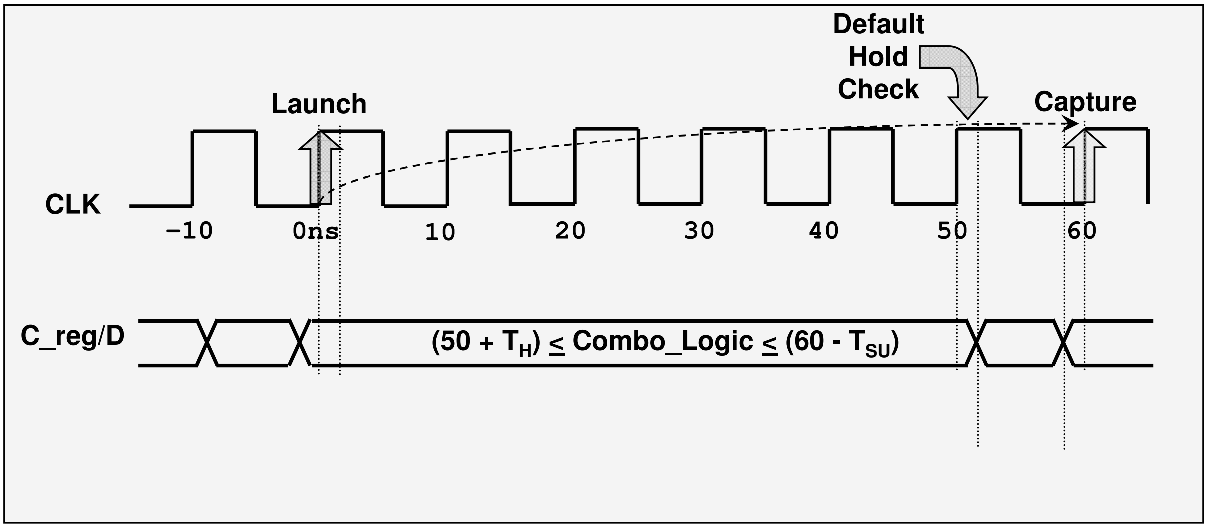
T_{SU} = setup time
T_H = hold time

The default setup multiplier value is 1 – this is why DC performs single-cycle optimization and analysis by default. By changing the setup muliplier to 6, DC will perform the setup analysis on edge 6, i.e. at 60 ns. This will allow the adder's logic to have a delay of (60 – setup_time – uncertainty).

Since multi-cycle path commands should be applied to the design prior to the first compile, you can not use library cell pin names as your start or endpoints. For registers, it is generally possible to use the register's *cell name* as a start or end-point – DC will automatically find the correct –*from* and –*to* pins. If this is too ambiguous (e.g. multiple startpoint pins: *clk* and *enable*, or multiple endpoint pins: *data*, *set* and *reset*), use the following for –*from* or *-to*: `[get_pins XYX_reg/gtech_pin_name],` where `gtech_pin_name` is equal to `clocked_on` for the clock pin or `next_state` for the data input pin. Note that after synthesis the *gtech* pin names are replaced by the actual *library cell pin* names.

Default Hold Check

```
set_multicycle_path -setup 6 -to C_reg[*]
```



Why is hold check performed at 50 ns?

10-19

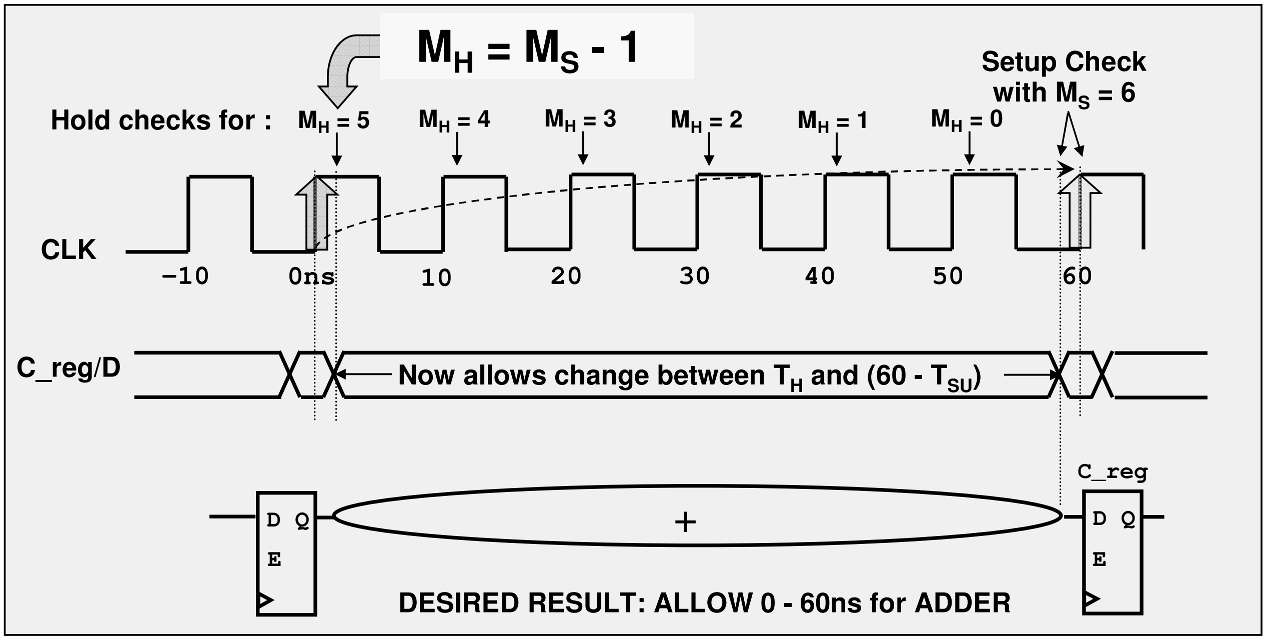
The default HOLD multiplier is set to 0. It is not intuitively obvious, but this means that the Hold check is always performed in the same cycle as the setup check, namely on the clock edge before the setup check.

DC assumes that the clock edges at 10/20/30/40/50 ns can cause metastability if they occur at the same time the data changes. Having the hold check at 50 ns avoids any metastability issues.

However, synthesizing a path that has a setup requirement of 60 ns and a hold requirement of 50 ns is virtually impossible (consider worst case/best case PVT corner variations plus LSB versus MSB delay differences)!

Set the Proper Hold Constraint

```
set_multicycle_path -setup 6 -to C_reg[*]
set_multicycle_path -hold  5 -to C_reg[*]
```



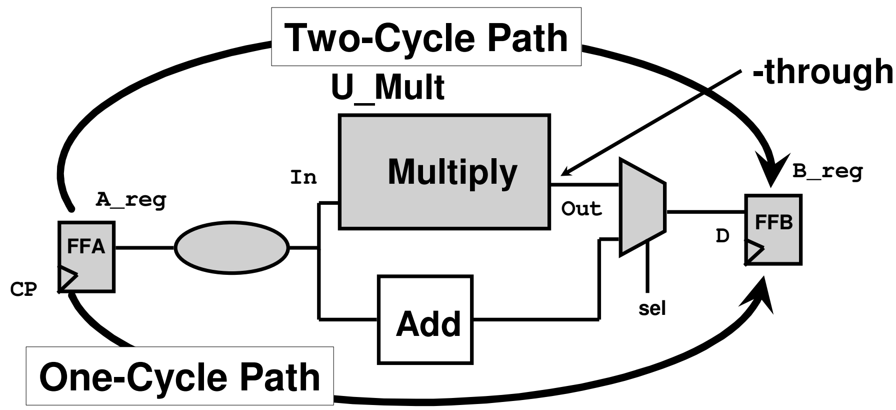
10- 20

M_H stands for Hold Multiplier, M_S for Setup Multiplier. The default hold-check edge is one clock edge before the setup-check edge.

By increasing the Setup multiplier from the default of 1, the clock edges at which the setup check as well as the hold check is performed are increased by that many clock cycles. Conversely, by increasing the Hold multiplier from the default value of 0, the edge at which the hold check is performed is *decreased* by that many clock cycles. So, if you wish to move ONLY the setup-check edge while keeping the hold-check edge in its original position, always set the hold multiplier to one less than the setup muliplier.

Another Example

```
set_multicycle_path -setup 2 -through U_Mult/Out
set_multicycle_path -hold 1 -through U_Mult/Out
```



10- 21

For situations when there are multiple paths that go through at a given point, you may need to specify a specific start and/or endpoint to isolate the path, for example:

```
set_multicycle_path -setup 2 -from A_reg/clocked_on \
   -through U_Mult/Out -to B_reg/next_state
set_multicycle_path -hold 1 -from A_reg/clocked_on \
   -through U_Mult/Out -to B_reg/next_state
```

Note: If the “Multiply” block is auto-ungrouped during optimization, the “timing exception” (multi-cycle or false-path) constraints are automatically “pushed down” to the appropriate elements inside the block, so the path remains correctly constrained.

Always Check for Invalid Exceptions

No warnings are issued if an invalid exception is applied to a design, so it is recommended to explicitly check for invalid exceptions:

```
report_timing_requirements -ignored
```

```
Description                                       Setup            Hold
---------------------------------------------------------------------------
NONEXISTENT PATH                                  FALSE            FALSE
        -from    { IO_PCI_CLK\
                   pclk }\
        -to      { IO_SDRAM_CLK\
                   SDRAM_CLK}
INVALID FROM OBJECT                                 6                  5
        -from    FF1/Q
```

To remove invalid exceptions:

```
reset_path -from FF1/Q
```

10- 22

There is no **report_timing_requirements –valid**! **report_timing_requirements** reports ALL exceptions, valid and invalid ones.

```
dc_shell-xg-t> report_timing_requirements  -help
Usage: report_timing_requirements    # report timing_requirements
        [-attributes]          (path timing attributes)
        [-ignored]             (ignored path timing attributes)
        [-from <from_list>]    (from clocks, cells, pins, or ports)
        [-through <through_list>]
                               (list of path through points)
        [-to <to_list>]        (to clocks, cells, pins, or ports)
        [-expanded]            (report exceptions in expanded format)
        [-nosplit]             (do not split lines when column fields overflow)
```

Multi-Path Constraints

Multiple Clocks - Synchronous

Multiple Clocks - Asynchronous

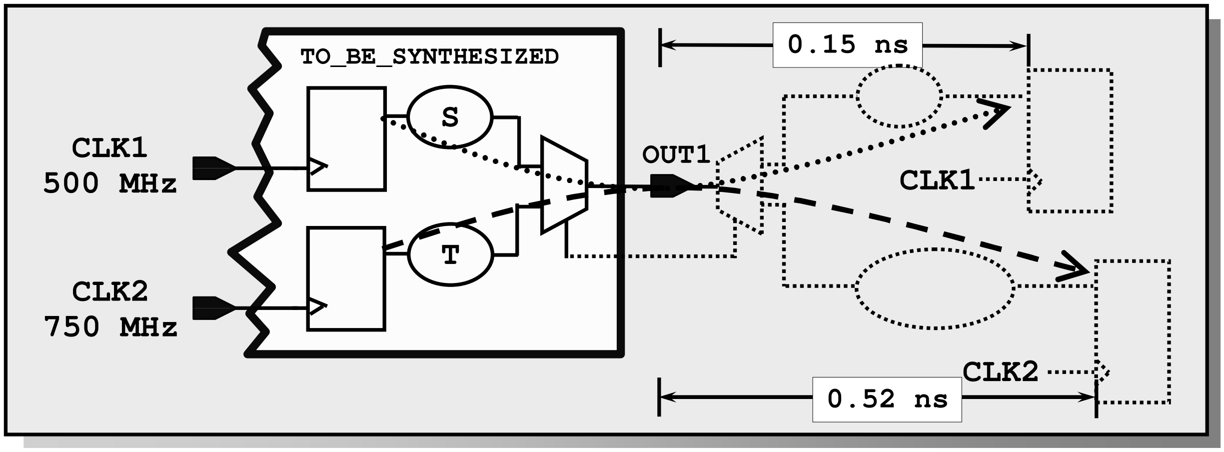
Multi-Cycle Path Constraints

Multi-Path Constraints

10- 23

Exercise: Multi-Path Constraints

The path through S is captured by the CLK1 register;
The path through T is captured by the CLK2 register.

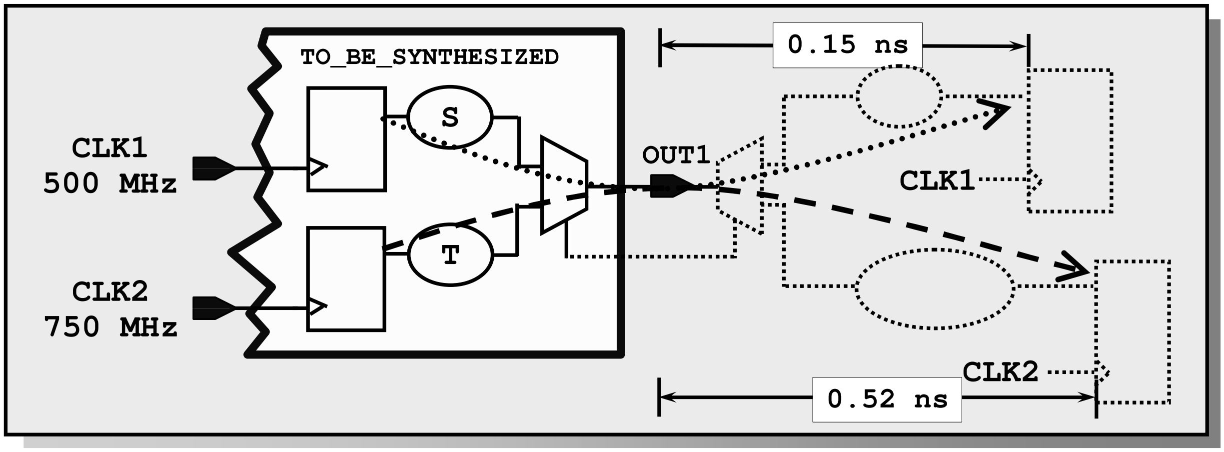


How do you constrain this design?

10- 24

Solution: Multi-Path Constraints (1 of 2)

```
create_clock -period 2.0 [get_ports CLK1]
create_clock -period [expr 1/0.75] [get_ports CLK2]
set_output_delay -max .15 -clock CLK1 [get_ports OUT1]
set_output_delay -max .52 -clock CLK2 -add_delay [get_ports OUT1]
```

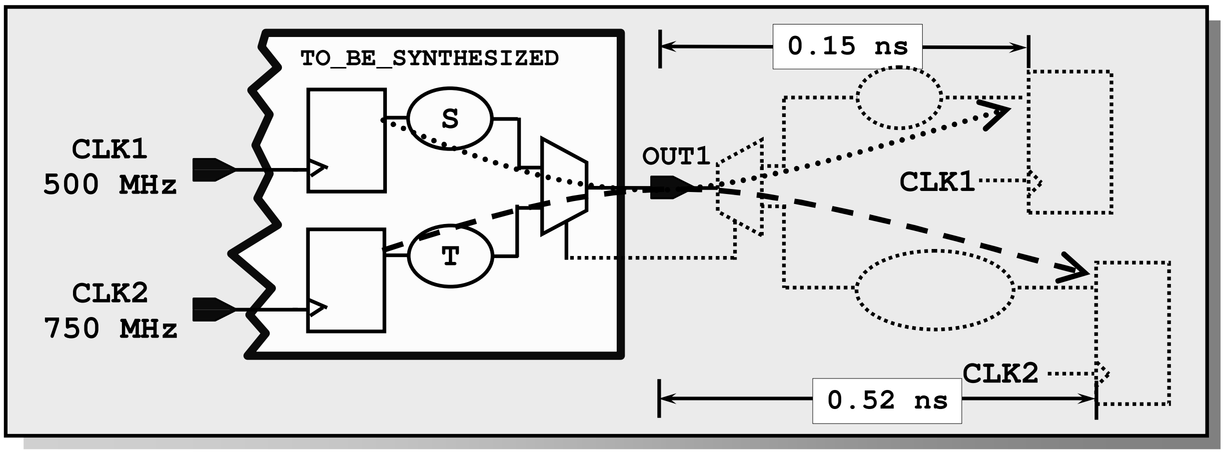


Is this enough?

10- 25

Solution: Multi-Path Constraints (2 of 2)

```
create_clock -period 2.0 [get_ports CLK1]
create_clock -period [expr 1/0.75] [get_ports CLK2]
set_output_delay -max .15 -clock CLK1 [get_ports OUT1]
set_output_delay -max .52 -clock CLK2 -add_delay [get_ports OUT1]
```



10-26

```
set_false_path -from [get_clocks CLK1] -to [get_clocks CLK2]
set_false_path -from [get_clocks CLK2] -to [get_clocks CLK1]
```

Or, if you need to be a little more specific and the register output signal names are *S* and *T*, respectively:

```
set_false_path -from S_reg -to [get_clocks CLK2]
set_false_path -from T_reg -to [get_clocks CLK1]
```

Summary: Commands Covered

mydesign.con

```
set_false_path -from [get_clocks CLKA] -to [get_clocks CLKB]

set_multicycle_path -setup 2 -from -from A_reg \
                    -through U_Mult/Out -to B_reg
set_multicycle_path -hold 1 -from -from A_reg \
                    -through U_Mult/Out -to B_reg

report_timing_requirements
report_timing_requirements -ignored
reset_path -from FF1/Q
```

10- 27

Summary: Unit Objectives

You should now be able to:

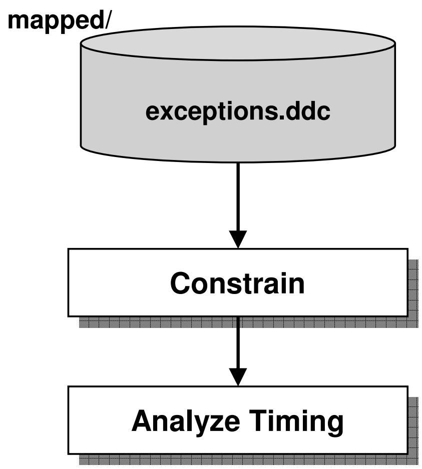
- Constrain a synchronous multi-clock design
- Constrain an asynchronous multi-clock design
- Constrain a multi-cycle design

10- 28

Lab 10: Multiple Clocks and Timing Exceptions

60 minutes

Constrain a multi-clock design using virtual clocks and timing exceptions



10- 29

This page was intentionally left blank.

Agenda

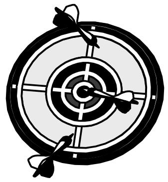
DAY 3

9	More Constraint Considerations (Lab cont'd)
10	Multiple Clock/Cycle Designs
11	Synthesis Techniques and Flows
12	Post-Synthesis Output Data
13	Conclusion

Synopsys 10-I-011-SSG-013

11- 1

Unit Objectives

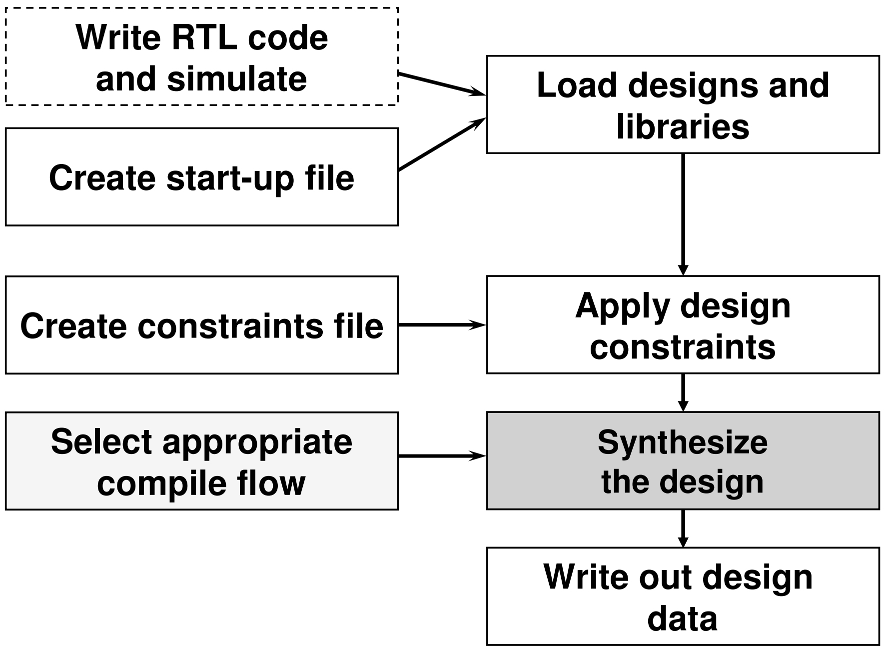


After completing this unit, you should be able to:

- **Describe five additional DC techniques for improving synthesis results - decide when they are applicable**
- **Select and execute the most appropriate *compile flow***
  - DC Ultra - Topographical
  - DC Ultra – WLM
  - DC Expert

11- 2

RTL Synthesis Flow



11- 3

Improving Results and Compile Times

- **Besides `compile` or `compile_ultra`, with their recommended options, there are additional techniques for improving the results of designs with:**
  - Arithmetic components
  - Pipelines
  - Poor hierarchical partitioning
  - Aggressive DRC requirements
  - Specific paths requiring more optimization focus
- **There is also the capability to perform *parallel synthesis* on large designs which require faster or frequent compile runs**

See Appendix 1
for more details

11- 4

Overview of Techniques

Techniques for improving synthesis results:

- Arithmetic components**: Use the *DesignWare* library**
- Pipelines**: Invoke the *register-repositioning* optimization**
- Poor partitioning**: Apply appropriate *auto-ungrouping* settings**
- Aggressive DRC requirements**: Modify the *optimization cost priority***
- Specific paths requiring more optimization focus**: Create *path groups* with *critical ranges* and *weights***

11- 5

Unit Agenda

Techniques for improving synthesis results in designs with:

- Arithmetic components: *DesignWare*
- Pipelines: *Register repositioning*
- Poor hierarchical partitioning: *Auto-ungrouping*
- Aggressive DRC requirements: *Cost priority*
- Specific paths requiring more optimization focus: *Path groups*

The recommended flows

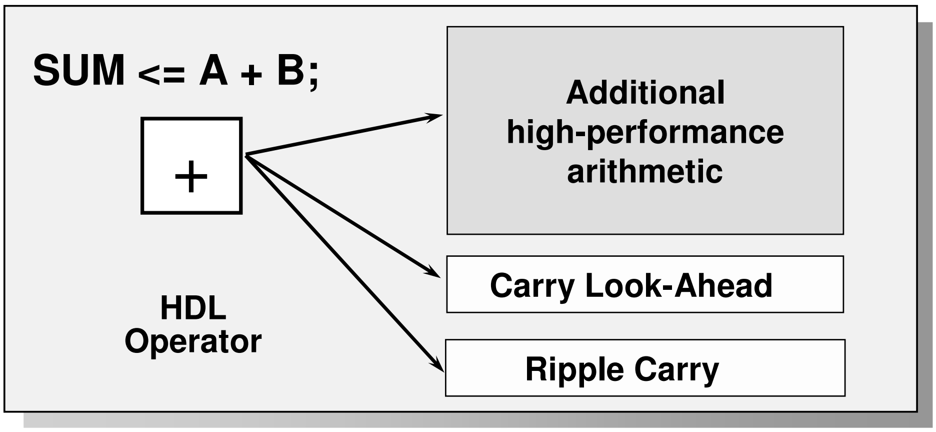
- Selecting the appropriate flow
- Detailed scripts of flows
  - DC Expert
  - DC Ultra – WLM
  - DC Ultra – Topographical

11- 6

Arithmetic Components

By default, DC makes available a sub-set of the possible architectures, or implementations, for arithmetic logic – this may not result in optimum QoR

? How can you take advantage of DC's full set of high-performance arithmetic implementations?



11-7

Answer: By accessing the DesignWare library

What is the *DesignWare* Library?

- **A huge collection of IP blocks and Datapath components:**
  - Technology independent, pre-verified, reusable, parameterizable, synthesizable
- **Accessing the Right Component:**
  - *Operator inferencing* for arithmetic operators
    - +, -, *, >, =, <
    - Operators greater than 4 bits wide infer a hierarchical sub-block
  - *Instantiation* for a wide variety of standard IP
    - DW_fifo_..., DW_shiftreg, DW_div_seq, DW_ram_...
- **Benefits of using the *DesignWare* Library:**
  - Better Quality of Results and Faster designs
  - Increased Productivity and Design Reusability
  - Decreased Design and Technology Risk
- **Requires a *DesignWare license* and two *variable* settings**

11- 8

Arithmetic components are automatically *inferred* by using their corresponding operators (+, -, *, >, =, <) in the RTL code. It is therefore very straightforward to infer very complex arithmetic circuits. All the other functional components (which do not have a standard RTL *symbol* like the arithmetic operators) are instantiated in the code – also very straightforward.

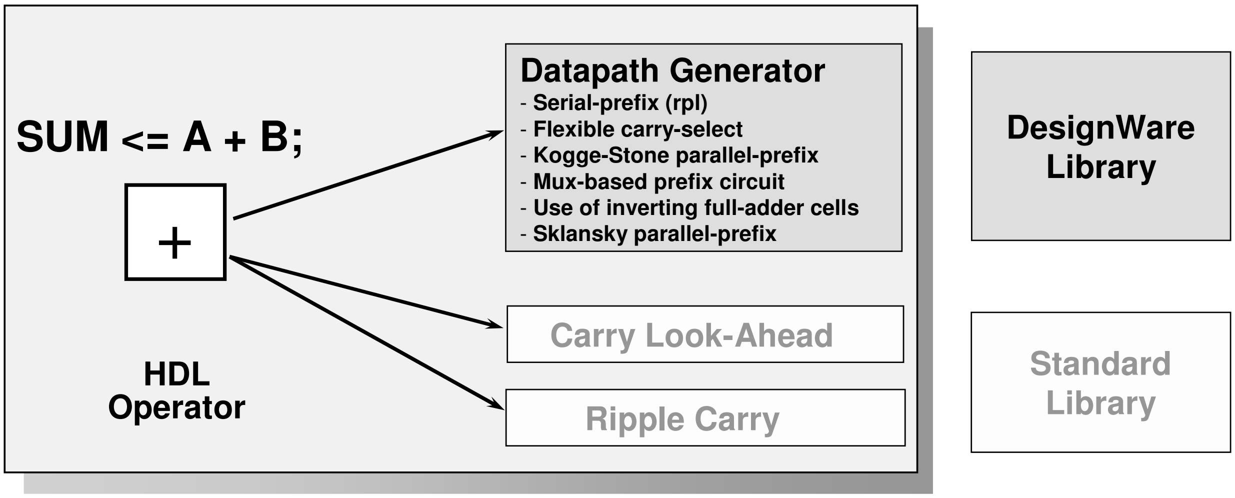
For datasheets of DesignWare components refer to the *DesignWare Library Databook*.

Basic architectures, or *implementations*, of +, -, *, >=, <, <= etc, are accessible by default from the 'standard' DesignWare library. To access the entire collection of basic and high-performance arithemetic implementations, plus the entire collection of additional standard IP, a DesignWare library license is required, and the `synthetic_library` and `link_library` variables need to be modified.

The DesignWare library resides in the directory: *$SYNOPSYS/libraries/syn*

Better QoR for Singleton Arithmetic

- **Multiple implementations for each operator allow DC to evaluate speed/area tradeoffs and choose the best**
  - The smallest that meets timing



- **Once selected, each implementation is then optimized for the target technology library**

11- 9

The *standard library* is automatically included, and set up for use with DC Expert. The *DesignWare library* requires a separate *DesignWare license*, as well as a couple of *variable* settings, to be enabled.

The datapath generator, which is included with the DesignWare license and enabled by default, dynamically builds the appropriate implementation (the smallest that meets timing) for each arithmetic operator (+, -, *) , rather than selecting an implementation from a “static” set of pre-built architectures (e.g. *carry look-ahead* and *ripple carry* from the *standard* library). For example, for an adder it uses a constraint-driven variable prefix architecture (implementation architecture name: *pparch*), which includes:

- Serial-prefix (rpl)
- Flexible carry-select
- Kogge-Stone parallel-prefix (like "fastcla")
- Mux-based prefix circuit
- Use of inverting full-adder cells
- Sklansky parallel-prefix

Enabling the DesignWare Library

- **You do not need to do anything to access the *Standard* library (*standard.sldb*)**
  - DC is set up to use this library by default
- **To enable use of the datapath generator and access to the entire collection of DesignWare IP, you must enable the *DesignWare license* and specify these variables[1]**

.synopsys_dc.setup

```
set target_library 65nm.db
set link_library "* $target_library"
set search_path "$search_path mapped rtl libs cons"

# These variables are automatically set if you perform Ultra optimization
# Specify for use during optimization
set synthetic_library dw_foundation.sldb
# Specify for cell resolution during link
set link_library "$link_library $synthetic_library"
```

11-10

[1] When executing `compile_ultra` or a `compile` with Ultra optimization enabled, these variables are automatically set for you. If you are executing non-Ultra compile (regular `compile` command), then you must explicitly set the variables above before compiling.

DesignWare Recommendations

The DesignWare datapath generator helps to achieve faster and smaller arithmetic logic. The IP collection increases productivity and reduces risk.

If you have a DesignWare license - Use it!

License required for DC Ultra (`compile_ultra`)

(no need to set variables)

License recommended for DC Expert (`compile`)

```
(set synthetic_library dw_foundation.sldb
set link_library "$link_library $synthetic_library")
```

11- 11

Unit Agenda

Techniques for improving synthesis results in designs with:

- Arithmetic components: *DesignWare*
- Pipelines: *Register repositioning*
- Poor hierarchical partitioning: *Auto-ungrouping*
- Aggressive DRC requirements: *Cost priority*
- Specific paths requiring more optimization focus: *Path groups*

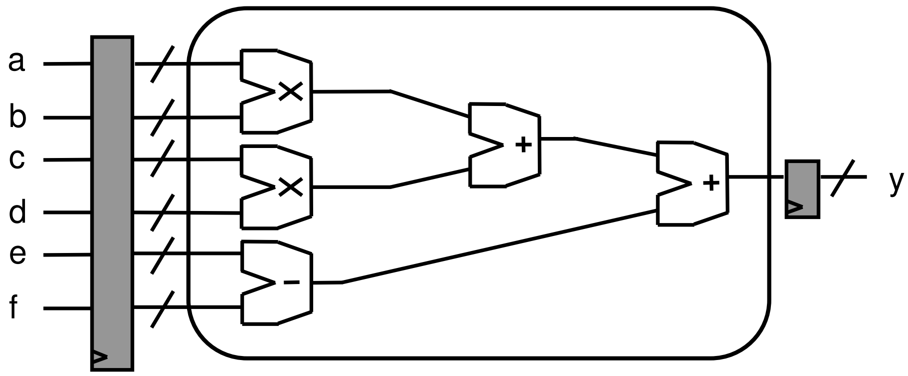
The recommended flows

- Selecting the appropriate flow
- Detailed scripts of flows
  - DC Expert
  - DC Ultra – WLM
  - DC Ultra – Topographical

11- 12

The problem: Large Delay between Registers

```
y <= a*b + c*d + e-f;
```

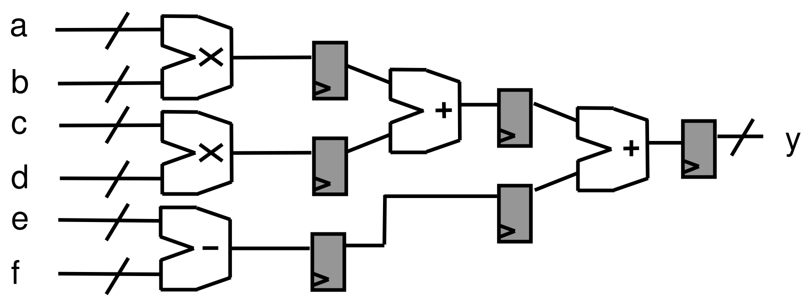


How can we speed up the clock frequency that this design can run at?

11- 13

Some Things to Try ...

- **Add pipeline stages**
- **Use the best available algorithms to optimize the arithmetic logic: `compile_ultra`**



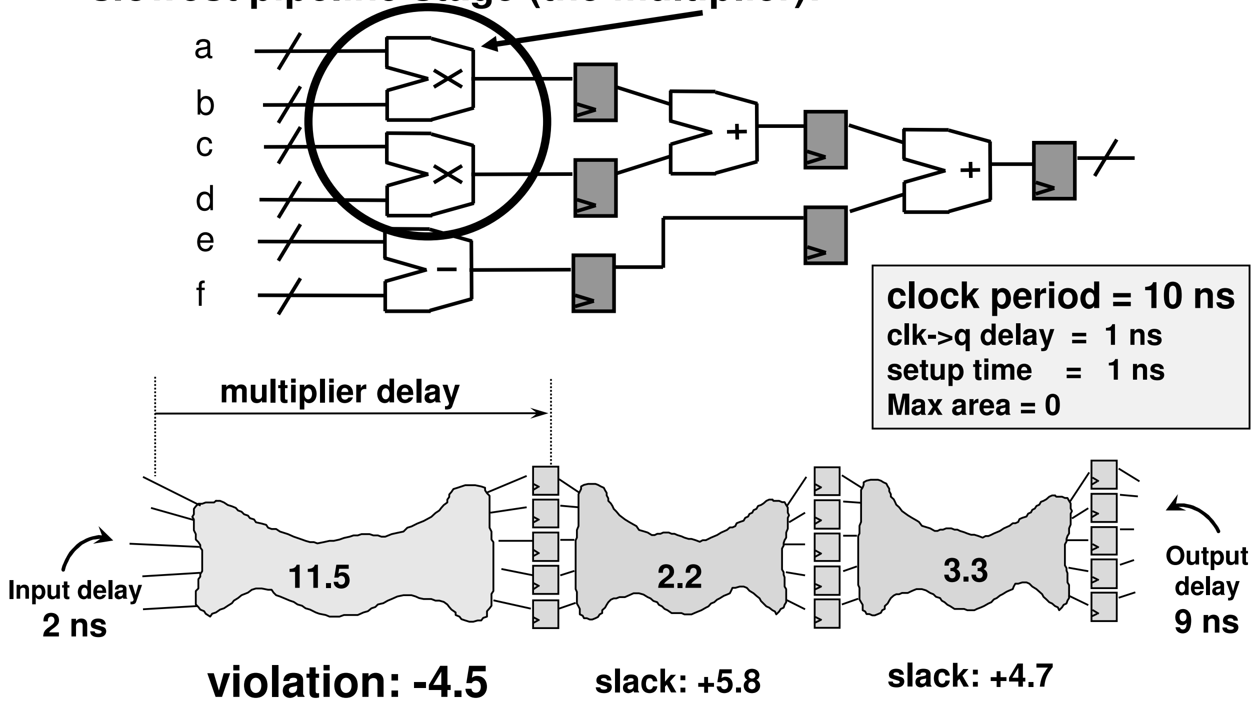

```
if (clk'event and clk='1') then
begin
    prodAB <= a * b;
    prodCD <= c * d;
    diffEF <= e - f;

    p2_1   <= prodAB + prodCD;
    p2_2   <= diffEF;

    y      <= p2_1 + p2_2 ;
```

```
always @ (posedge clk)
begin
    prodAB <= a * b;
    prodCD <= c * d;
    diffEF <= e - f;

    p2_1   <= prodAB + prodCD;
    p2_2   <= diffEF;

    y      <= p2_1 + p2_2 ;
```

11- 14

What if the Pipeline is Still Violating Timing?

The clock frequency is limited by the total delay in the slowest pipeline stage (the multiplier):



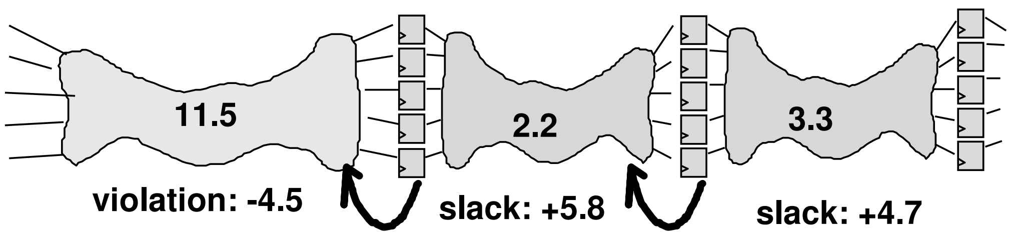
11-15

The above circuit will only work with clock period >= 14.5ns instead of the required 10ns.

The Solution: Register Repositioning[1]

```
optimize_registers
```

- Register repositioning moves registers to borrow slack from ‘positive slack’ stages and help violating stages
- Requires an Ultra license
- Only works on a mapped or previously-compiled design

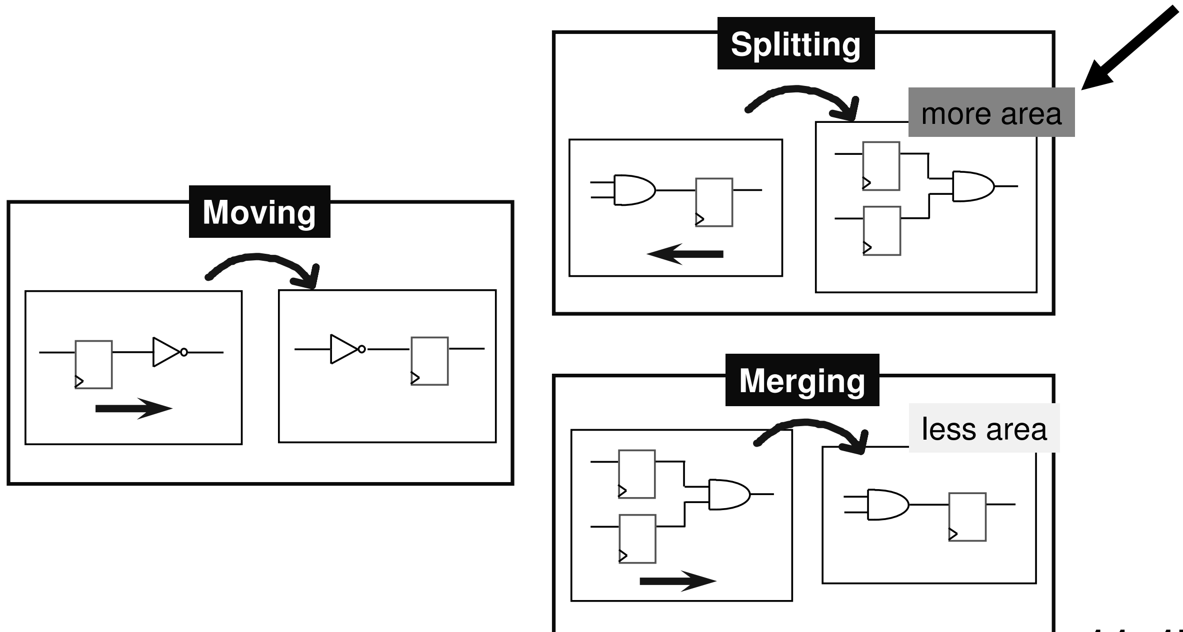


11-16

1) Register Repositioning is also referred to as:
Register Retiming, or
Pipeline Retiming, or
Behavioral Retiming (BRT)

How Does Register Repositioning Work?

- **Register repositioning works by moving, splitting, or merging registers through the cones of logic**
  - "End-to-End" functionality of the circuit is unchanged

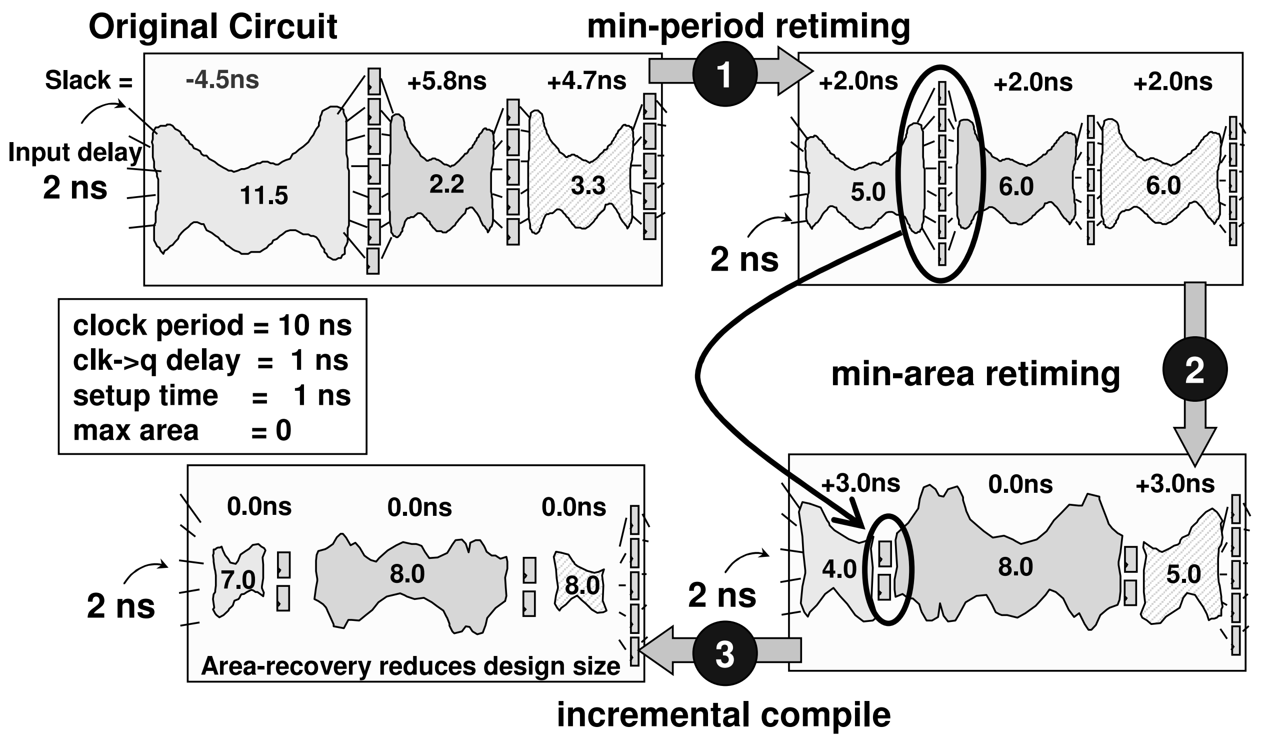


11- 17

Notice that the overall register count may increase, or decrease, but the number of stages remains the same. As in the case of merging, the register count (and overall area) may decrease, but other times (like in the case of splitting) register count (and overall area) may increase. It usually depends on whether registers are moving from right to left (toward wider points in the fanin cone) or left to right (toward narrower points in the fanin cone). Although no register stages are being added, registers may have to be added to capture the larger number of intermediate nodes at wider points in the fanin cone (left side) and preserve the same overall latency.

DC (optimize_registers) makes intelligent choices as to whether a given register should be repositioned or not in order to best meet constraints.

What `optimize_registers` does



Resulting design meets timing while the area is minimized!

11- 18

Phase I: Register Moving
- Moves registers across combinational elements
- Maintains existing hierarchy – registers can move across hierarchical boundaries, including DesignWare hierarchy
- Does not modify any combinational element
- Does minimum period retiming (regardless of area)
- Does minimum area retiming (following minimum period retiming)

Phase II: Final Optimization
- Incremental Compile with Boundary Optimization
- Optimizes combinational elements between registers to further reduce delay violations or meet area goal
- Does not move any sequential element

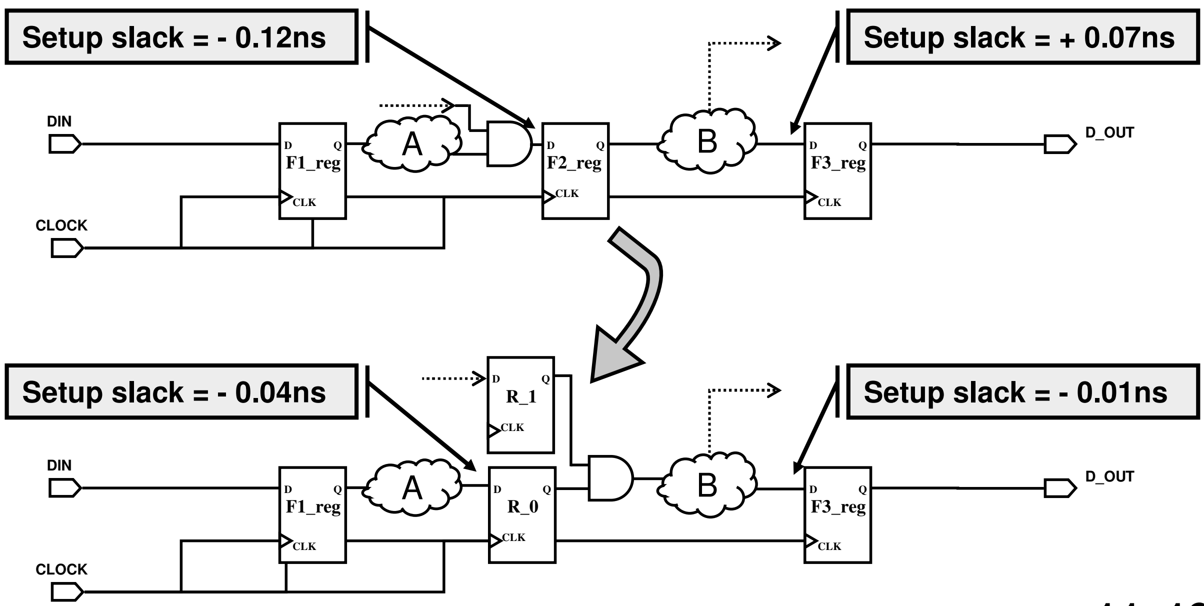
Overall input-to-output functionality remains unchanged

Requires an Ultra license

Note: A file called *default.svf* is created automatically during compile, which records the "changes" that *boundary optimization*, *register repositioning* and *ungrouping* make to the design. This file is readable by Formality, Synopsys' formal verification or equivalency checking tool. To rename the file use `set_svf <My_name.svf>`; invoke this command before reading in RTL (can put it in your *.synopsys_dc.setup* file). If using a third-party formal verification or equivalency checker try using `set_vsdc <My_name.vsdc>` to write out an ASCII "vsdc" file, which is intended to be readable by third party tools. Since this is a relatively new command and format the file may not be readable by all formal verification tools.

What does `compile_ultra -retime` do?

The `-retime` option invokes local 'adaptive retiming' optimization, which is intended to reduce critical timing violations (WNS) in non-pipeline register paths



11-19

In the example above, adaptive retiming is able to reduce the WNS from -0.12ns to -0.04ns. It introduces a new, but small violation of -0.01ns to accomplish this.

WNS = Worst negative slack

A pipelined design is a design with a logic-register-logic-register-logic…. architecture in which the logic values of the intermediate sequential or combinational logic is not "tapped off" or otherwise required to be known or fixed. The only requirement is that the pipeline is fed some input signals, and some known cycles later (the latency), known logic values appear at the OUTPUT of the pipeline – only the logic definition of the outputs must remain fixed.

In this context, a non-pipelined design is a design that also contains logic-register-logic-register circuitry, but intermediate logic may be tapped off in some places along the data path.

Adaptive Retiming Details

- **Performs "local" optimization - only on potential timing-critical paths**
  - No "global" min-period retiming
- **No runtime overhead**
- **May increase register count due to "splitting"**
  - No min-area retiming
- **Moved registers are renamed "R_##"**
  - No way to relate back to original register name[1]
  - Prevent retiming on specific registers or sub-designs with `set_dont_retime <cells or designs> true`

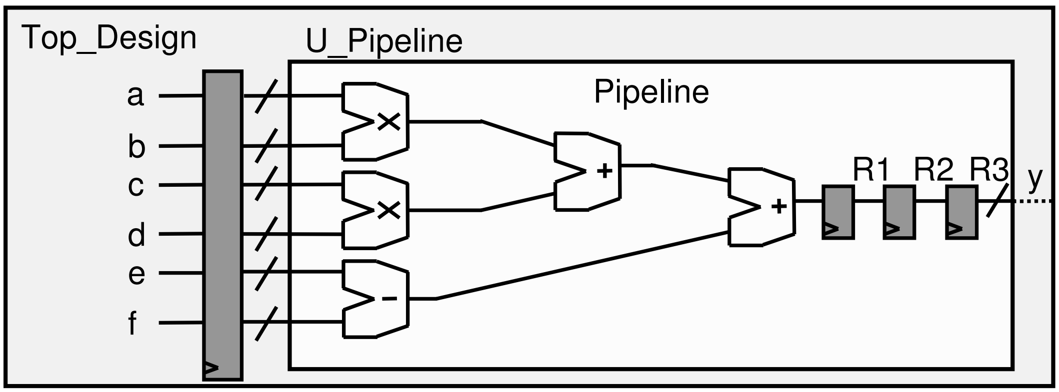
Do not use `-retime` if your design requires <u>fixed</u> non-pipeline register <u>positions</u> and/or <u>names</u>!

11-20

[1] Renamed registers will be included in the *default.svf* file described earlier, which records netlist changes that are intended to be read by Synopsys' formal verification tool, *Formality*.

Pipeline Assumptions and Recommendations

- **Pipeline design assumptions**
  - Pipeline circuitry may co-exist with non-pipelined circuitry
  - Pipeline logic can be arithmetic or non-arithmetic (random logic)
- **RTL recommendations**
  - Put each pipeline in its own sub-block
  - Group all pipeline registers at input or output of pipeline
- **Compile recommendations**
  - Perform the initial compile (`compile_ultra -retime ..`) with relaxed pipeline timing – map to gates with minimal optimization



11-21

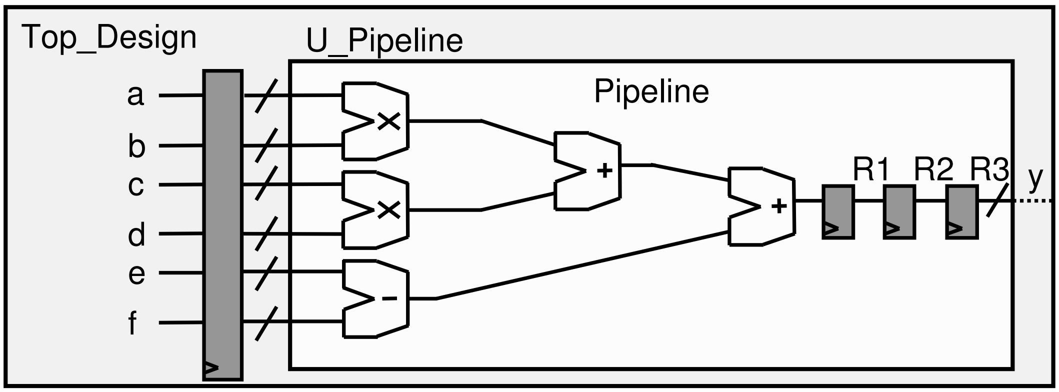
Since most designs contain both pipelined and non-pipelined circuitry, it is recommended to separate the pipelined circuitry by placing it in its own sub-design(s). This makes it easier to apply `optimize_registers` only on the pipelined circuitry, without affecting the non-pipelined portion of the design. Conversely, this also allows the non-pipelined circuitry to be initially optimized, using `-retime`, without spending too many optimization cycles on the pipelined logic prior to invoking `optimize_registers`.

For best results it is recommended to initially group the pipeline registers together at the input or output of the pipeline in the RTL code, rather than interspersing them among the logic (this is not a requirement – just recommended when possible). See example code at the bottom of the next page.

If the register are grouped together it is further recommended to relax the timing constraints for the pipeline's datapath or combo logic for the first compile using `set_multicycle_path`. The first compile simply converts the pipeline GTECH logic to gates without much optimization, in preparation for `optimize_registers`, while optimizing the non-pipeline portions of the design using adaptive retiming (`-retime`). If the multi-cycle path is not applied then the pipeline logic will most likely be the critical path in the design, which will force DC to focus all its optimization cycles on this part of the design, while ignoring the rest of the design. The multi-cycle path must be removed after the first compile so that register repositioning sees and optimizes the actual violations.

Register Repositioning Flow Example

```
current_design Top_Design
        # Relax pipeline timing requirement for initial compile 1
set_multicycle_path -setup 3 -to U_Pipeline/R1_reg*
        # First compile with adaptive retiming for non-pipeline registers
compile_ultra -scan -retime -timing
        # Reset pipeline timing back to the original constraints 2
reset_path -to U_Pipeline/R1_reg*
        # Continue if pipeline violates timing; Skip if no pipeline issues: 3
set_optimize_registers true -design Pipeline
optimize_registers -only_attributed_designs
```



11-22

1) The default single-cycle timing should be relaxed to match the number of stages of the pipeline. If the pipeline will end up with registered inputs or outputs, then the number of stages equals the number of register banks in the design; If the pipeline is allowed to have combinational logic at both its inputs as well as outputs then the number of stages equals the number of register banks + 1. Register instance names are always in the form of OUTPUT_SIGNAL_NAME_**reg**; It is acceptable to specify the register cell name as a start or end point for timing exceptions. If necessary, use the "gtech" pin name ***next_state*** for the data pin and ***clocked_on*** for the clock pin.
2) After the first *compile* the multi-cycle timing should be removed to allow the "true" timing to be seen by optimize registers. Since the registers are now mapped you will need to find out the specific mapped register's clock pin or, as in this example, data pin name – in the example above this was assumed to be "d".
3) By setting the `set_optimize_registers` attribute to *true* on the pipeline sub-design only, and including the `-only_attributed_designs` option with `optimize_registers` you are ensuring that only the pipelined circuitry will be affected by register repositioning, while the remaining non-pipelined majority of the design is unaffected by this step.

Recommended RTL coding style for pipelined designs:

```
data <= a*b + c*d + e-f;

if (clk'event and
clk=`1') then
begin
    R1 <= data;
    R2 <= R1;
    R3 <= R2;
```

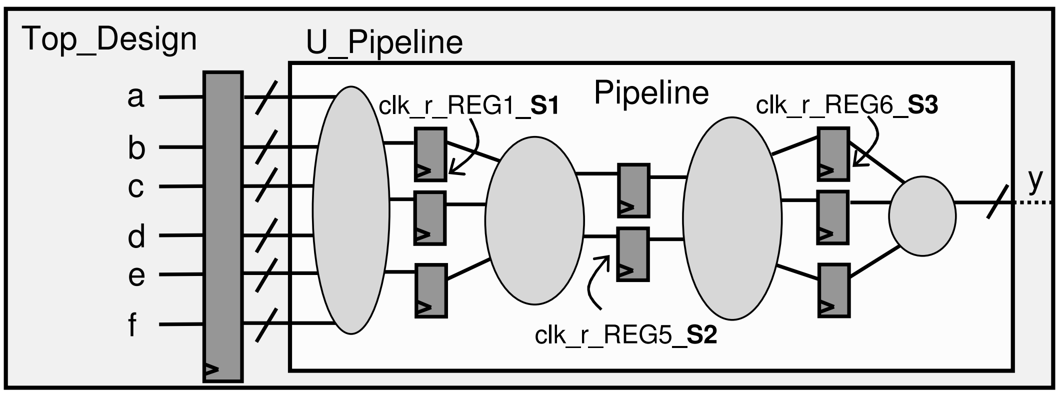
```
data <= a*b + c*d + e-f;

always @ (posedge clk)
begin
    R1 <= data;
    R2 <= R1;
    R3 <= R2;
```

Resulting Pipeline

- **All registers, including the output registers, can be repositioned**
- **Repositioned register cells are renamed:**

  XYZ_reg[#] → clockname_r_REG#_**S#**

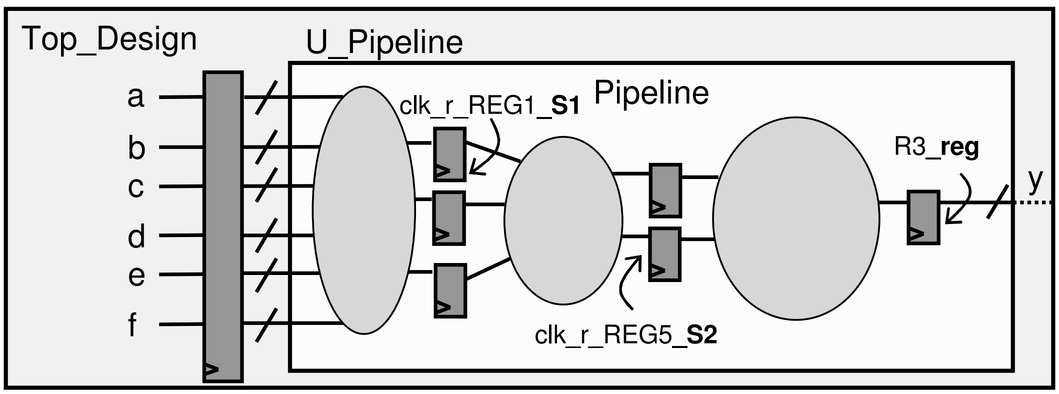
  where the ending **S#** represents the register stage number



11- 23

Maintaining Registered Outputs

```
current_design Top_Design
        # Relax pipeline timing requirement for initial compile
set_multicycle_path -setup 3 -to U_Pipeline/R1_reg*
        # First compile
compile_ultra -scan -retime -timing
        # Reset pipeline timing back to the original constraints
reset_path -to U_Pipeline/R1_reg*
        # Continue if pipeline violates timing; Skip if no pipeline issues:
set_dont_touch [get_cells U_Pipeline/R3_reg*] true
set_optimize_registers true -design Pipeline
optimize_registers -only_attributed_designs
```



11-24

Register Repositioning Recommendations

If your design is timing-critical and non-pipeline registers are allowed to be re-positioned, invoke adaptive retiming

```
compile_ultra -scan -retime -timing
```

If you have pipelined sub-design(s) invoke register repositioning to optimize your pipeline

```
optimize_registers
```

Recommended with DC Ultra

Not available with DC Expert

11- 25

Unit Agenda

Techniques for improving synthesis results in designs with:

- Arithmetic components: *DesignWare*
- Pipelines: *Register repositioning*
- Poor hierarchical partitioning: *Auto-ungrouping*
- Aggressive DRC requirements: *Cost priority*
- Specific paths requiring more optimization focus: *Path groups*

The recommended flows

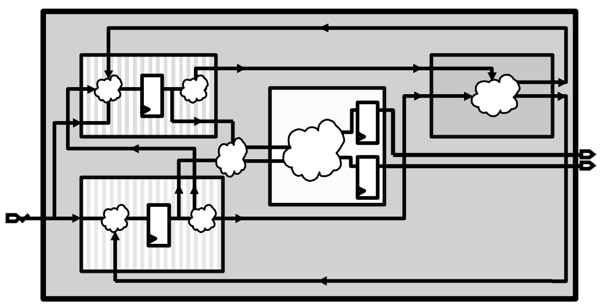
- Selecting the appropriate flow
- Detailed scripts of flows
  - DC Expert
  - DC Ultra – WLM
  - DC Ultra – Topographical

11- 26

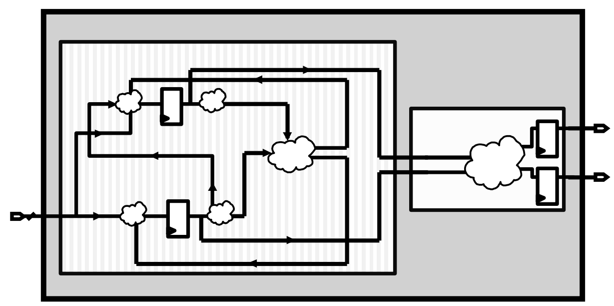
Partitioning Example 1: Sub-blocks

Your design's hierarchical partitioning was defined so that you can work in parallel to get to market faster, distributing the RTL coding to different teams based on block functions – not optimum for your integrated top-down synthesis.

RTL partitioning



Optimum partitioning

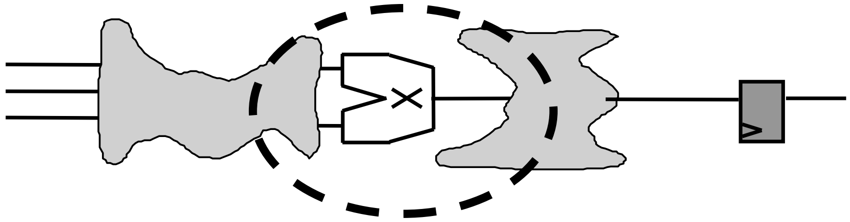


11- 27

See question on next page …

Partitioning Example 2: DesignWare Parts

Your design contains arithmetic logic (>4 bits) surrounded by random combinational logic. *DesignWare* will create a hierarchical sub-block for your arithmetic logic – not optimum partitioning.



How can you re-partition a design for optimum synthesis results after reading in RTL?

You can repartition *manually* <u>prior</u> to compiling with `group/ungroup` , or ...

11-28

Note: In the DC Expert flow you should always `compile` ... + `compile -incr` ... first before manually ungrouping arithmetic DesignWare parts. This allows DC to initially select, and possibly re-select the best *implementation* for the arithmetic logic. Once ungrouped, the implementation can not be changed.

Ungrouping During *Compile*

- **During *compile*, *auto-ungrouping* can automatically make “smart” ungrouping choices [1]:**

```
compile_ultra; # auto-ungrouping enabled
               # by default

compile -auto_ungroup area|delay
```

  - Ungrouping controlled through variables (discussed next)
  - To report designs auto-ungrouped during synthesis use **`report_auto_ungroup`**

- **Or, the <u>entire</u> hierarchy can be ungrouped [2]**

```
compile -ungroup_all
```

11- 29

[1] `compile -auto_ungroup area|delay` can be used in the “pseudo-Ultra” flow, if you have
an Ultra license but no DesignWare license, and hence can not execute `compile_ultra`.
Note that `compile -auto_ungroup` will NOT ungroup *DesignWare* parts – these must be ungrouped manually
`compile_ultra` performs auto-ungrouping by default (including most DesignWare parts[3]) .
Auto-ungrouping is discussed further in the next few slides.

[2] All sub-designs will be ungrouped, including most DesignWare parts[3], unless they are marked with a “dont_touch” attribute (`set_dont_touch`) .

[3] Except for special pipelined components, such as *DW02_mult_x_stage* and *DW_div_pipe*.

Note: A file called *default.svf* is created automatically during compile, which records the "changes" that *boundary optimization*, *register repositioning* and *ungrouping* make to the design. This file is readable by Formality, Synopsys’ formal verification or equivalency checking tool. To rename the file use `set_svf <My_name.svf>`; invoke this command before reading in RTL (can put it in your *.synopsys_dc.setup* file). If using a third-party formal verification or equivalency checker try using `set_vsdc <My_name.vsdc>` to write out an ASCII "vsdc" file, which is intended to be readable by third party tools. Since this is a relatively new command and format the file may not be readable by all formal verification tools.

Auto-Ungrouping with Ultra

- **Two types of *auto-ungrouping* can be performed during synthesis (requires an Ultra license):**
  - Delay-based: Only poorly-partitioned[1] sub-blocks (and their parent blocks) which cut through *violating timing paths* are considered for ungrouping
  - Area-based: All poorly partitioned sub-blocks (and their parent blocks) are considered for ungrouping
- ***Delay-* or *area-based* auto-ungrouping can be selected with `compile -auto_ungroup delay|area`**

See Appendix 2 for more details

- ***Delay-based* auto-ungrouping is automatically invoked with `compile_ultra` (even with the `-area` option)**
  - Initially performs *gtech-based* area-ungrouping[2]
- **Auto-ungrouping can be further controlled by *variables***

11-30

[1] A poorly partitioned sub-block is a hierarchical block (an instantiated module or entity in RTL, or a DesignWare part) which separates external and internal clouds of combinational logic at its input(s) and/or output(s), thereby preventing DC from being able optimize the combinational logic surrounding the hierarchy borders; It can also be a hierarchical block which separates combinational logic from the data input pin of a register, preventing DC from being able to perform *sequential optimization*.

[2] `compile_ultra` performs “gtech” area-based ungrouping prior to phase 1 of compile. This is based on the GTECH size of the sub-blocks: Very small sub-blocks are ungrouped independent of their partitioning or constraints. The block size limit for this “gtech” area-based ungrouping is not user-controllable. The area-based auto-ungrouping mentioned on the slide above refers to ungrouping which occurs during gate-level optimization, and is controllable (the size limit) by the user.

Note: Auto-ungrouping is not available during an *incremental* compile using `compile` **`-incremental`** `-map high -scan`.

Report auto-ungrouped designs with **`report_auto_ungroup`**.

Controlling Auto-Ungrouping

```
# Only sub-blocks containing max_limit number of
# cells (instances), or less, will be considered for ungrouping
set compile_auto_ungroup_delay_num_cells <max_limit>
set compile_auto_ungroup_area_num_cells <max_limit>
# Prevent parent-cells from being ungrouped while the culprit
# child cells which do not meet the num_cells limit remain grouped
set compile_auto_ungroup_count_leaf_cells true
# Ensure that sub-blocks with WLMs that are different
# than their parent cell are allowed to be ungrouped
set compile_auto_ungroup_override_wlm true
# Prevent glue logic at the top level of your current design
# or maintain the hierarchy of key blocks, e.g. for verification, pipeline designs
set_ungroup <top_level_or_other_key_cells> false
```

To achieve the best synthesis results set `max_limit` to 'infinite', e.g. 99,999,999

- May lose significant hierarchy in the design – consider post-synthesis simulation and physical layout affects
- Apply smaller limit if needed or control with `set_ungroup false`

11- 31

`compile_auto_ungroup_delay/area_num_cells` specifies, by default, the maximum number of top-level-only cells (instances) that a sub-block can have to be considered for delay- or area-based auto-ungrouping. The defaults are 500 and 30, respectively, which means that blocks containing 501, or 31, or more cells will never be ungrouped during delay/area-based auto-ungrouping, respectively. Ungrouping begins at the top-most sub-blocks and continues recursively down the hierarchy until the limit of 500/30 is reached, at which point the blocks containing 501/31 or more instances are not ungrouped. This means that, by default, many higher-level sub-blocks may be ungrouped (if they only contain instances of other sub-blocks, but no glue logic), while the lower-level cells, which actually contain the combinational logic that is causing the problem, remain grouped. It is also possible that you may end up with "glue logic" at the top level. `compile_ultra` performs only delay-based auto-ungrouping (ignores the `area_num_cells` variable), even with the `-area` option. Area-based auto-ungrouping can only be invoked with `compile -auto_ungroup area`. (see DC "Pseudo-Ultra" flow in the Appendix).

`compile_auto_ungroup_count_leaf_cells` directs DC to consider a parent cell for auto-ungrouping only if its cell count, including the cell-count of all of its child cells recursively down to the leaf cells, is at or below the *delay/area_num_cells* limit. By default, the cell count of a block is counted only as the number of cells instantiated at the top level of that block. It is recommended to always set this variable to true to prevent parent cells from being ungrouped while the actual child cells that are causing the problem are not (because the child cells exceed the *num_cells* limit).

`compile_auto_ungroup_override_wlm` allows sub-blocks with different Wireload Models (WLMs) than their parent cell to be considered for auto-ungrouping. By default, only blocks with the same WLM as their parent cell are considered. This variable should be set to *true* if the library has automatic wire-load selection enabled, or if the user manually applies multiple WLMs to different sub-blocks. If only one WLM is applied to the design then a *true* value for this variable has no effect on auto-ungrouping, so it is recommended to always set is to true.

`set_ungroup false` can be used to disable auto-ungrouping on specified cells. By default any cell that meets the above criteria can be auto-ungrouped, including top-level blocks. For example, if you want to avoid the possibility of glue logic at the top level you can disable auto-ungrouping on all top-level cells; You can also use this attribute to maintain the hierarchy of certain sub-blocks for verification purposes.

Design Partitioning Recommendations

If the RTL code is not partitioned optimally, perform manual or automatic repartitioning

DC Ultra: Apply the recommended auto-ungrouping control settings for `compile_ultra` or `compile -auto_ungroup`

DC Expert: Use `group/ungroup` or `compile -ungroup_all`

11-32

Unit Agenda

Techniques for improving synthesis results in designs with:

- Arithmetic components: *DesignWare*
- Pipelines: *Register repositioning*
- Poor hierarchical partitioning: *Auto-ungrouping*
- Aggressive DRC requirements: *Cost priority*
- Specific paths requiring more optimization focus: *Path groups*

The recommended flows

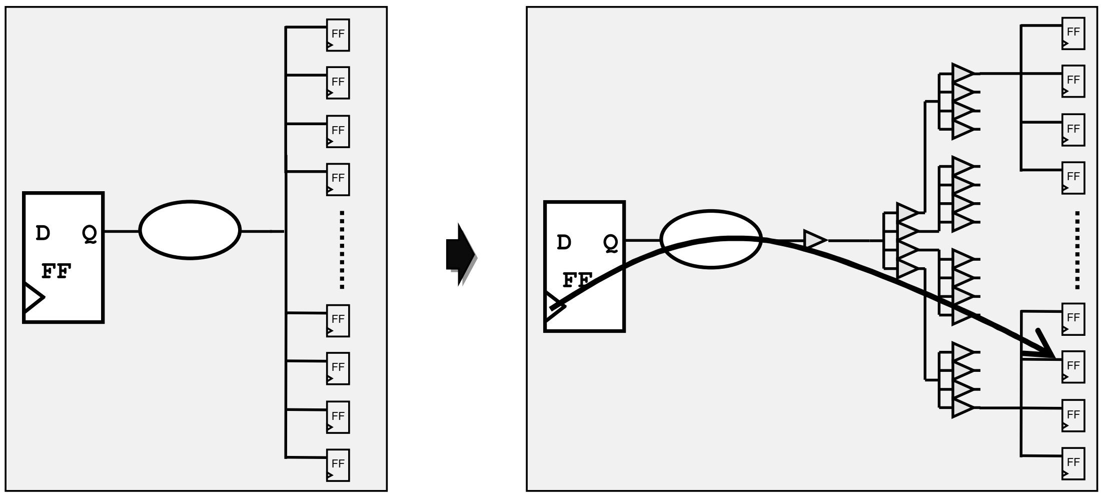
- Selecting the appropriate flow
- Detailed scripts of flows
  - DC Expert
  - DC Ultra – WLM
  - DC Ultra – Topographical

11- 33

The Problem: *DRC* versus Delay

DC's buffering or *DRC* rules ensure that your design meets maximum driver load and output transition time requirements - if necessary, at the expense of delay

How can you improve delay amid aggressive DRCs (and handle remaining DRC violations post-synthesis)?



11- 34

ANSWER: By applying a command to increase the *cost priority* of delay above the DRC cost priority.

DRC rules are usually included in the technology library in the form of `max/min_capacitance`, `max_transition` and/or `max_fanout` rules applied to the output pin(s) of each library cell. By default, DRC rules have a higher *cost priority* over delay. This means that DC will first attempt to meet all DRC rules without hurting delay, but if this is not possible, DC will try to meet the more aggressive DRCs even if this creates larger delays along critical paths. By increasing the cost priority of delay, DC will still attempt to fix DRC violations, but not at the expense of delay. Any remaining DRC violations can be addressed in the back-end or physical design.

The capacitance rules may be used to ensure that the load which each cell drives falls within the characterization range of the cell's timing model; Another use for limiting the `max_cap` is to control reliability effects of metal migration (large and prolonged current flows through thin wires will migrate metal electrons over time and cause narrow channels to thin out further, increasing the wire resistance and eventually causing an open circuit – reducing the capacitance reduces the overall "current energy" passing through the wire, which can increase reliability).

Controlling the transition time is commonly used as an indirect way to reduce transient power dissipation in the design. While an input signal of a CMOS gate is in transition (between 0.2 and 0.8xVDD) both the pull-up PMOS and the pull-down NMOS transistors are turned on, dissipating power directly from VDD to ground through the transistor pairs. By "sharpening" the transition time, the transient power dissipation time can be reduced, saving overall power dissipation.

Max fanout is a design rule which a vendor can use as they choose, to control DC's buffering with another concern in mind, in addition to the above ones. There is no pre-determined definition of what this is used for. Max fanout simply adds up the value of an attribute called `fanout_load`, which would be applied to the input pins of the cells which are connected to a certain driver, and compares the total `fanout_load` to the `max_fanout` rule, and optimizes the design to meet the rule. The vendor determines the meaning or "unit" of one `fanout_load`.

It is not recommended that the user changes or manipulates these design rules, unless they have a very clear understanding of the implications, as well as a good reason to do so.

DRC Priority Recommendation

Use `set_cost_priority -delay`

after the first-pass compile(s), if your design is still violating timing, and if it is acceptable to handle any remaining DRC violations post-synthesis

Automatically set with DC Ultra (`compile_ultra`)

Recommended with DC Expert (`compile`)

11-35

Unit Agenda

Techniques for improving synthesis results in designs with:

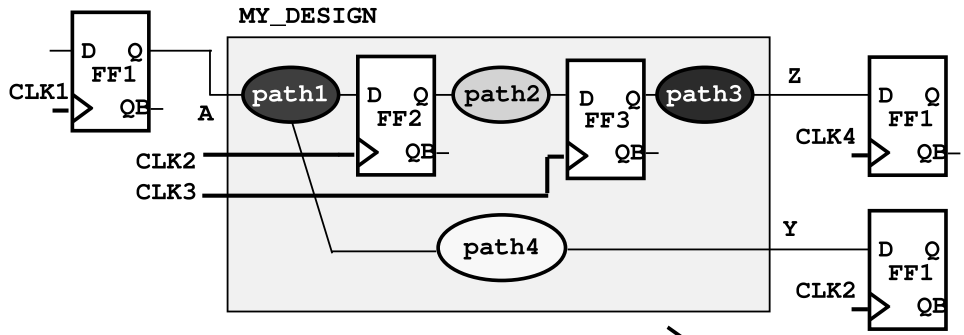
- Arithmetic components: *DesignWare*
- Pipelines: *Register repositioning*
- Poor hierarchical partitioning: *Auto-ungrouping*
- Aggressive DRC requirements: *Cost priority*
- Specific paths requiring more optimization focus: *Path groups*

The recommended flows

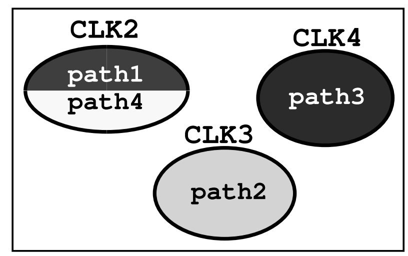
- Selecting the appropriate flow
- Detailed scripts of flows
  - DC Expert
  - DC Ultra – WLM
  - DC Ultra – Topographical

11-36

Recall: DC Organizes Paths into Groups



Timing Paths



Path Groups

- **Paths are grouped by the clocks controlling their <u>endpoints</u>**
- **DC optimizes each path group in turn, starting with the critical path in each group**

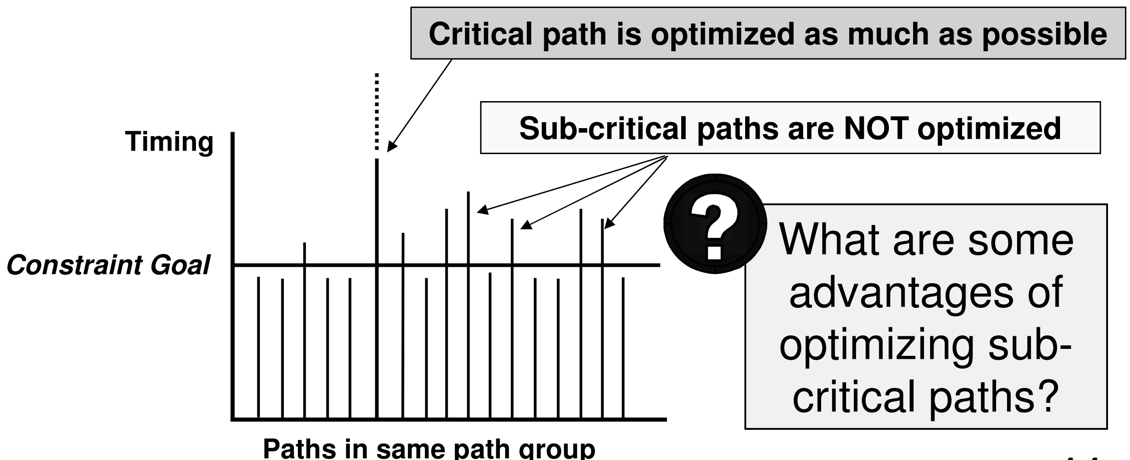
11- 37

Path groups are named the same as their related (end-point) clock object names, by default.

`report_path_group`: Reports the path groups which were defined in the current design.

Problem: Sub-Critical Paths Ignored

- **By default, optimization within a path group stops when:**
  - All paths in the group meet timing, or
  - DC cannot find a better optimization solution for the critical path – it reaches a point of diminishing returns
  - Sub-critical paths are not optimized - to save compile time!



What are some advantages of optimizing sub-critical paths?

11-38

ANSWER:

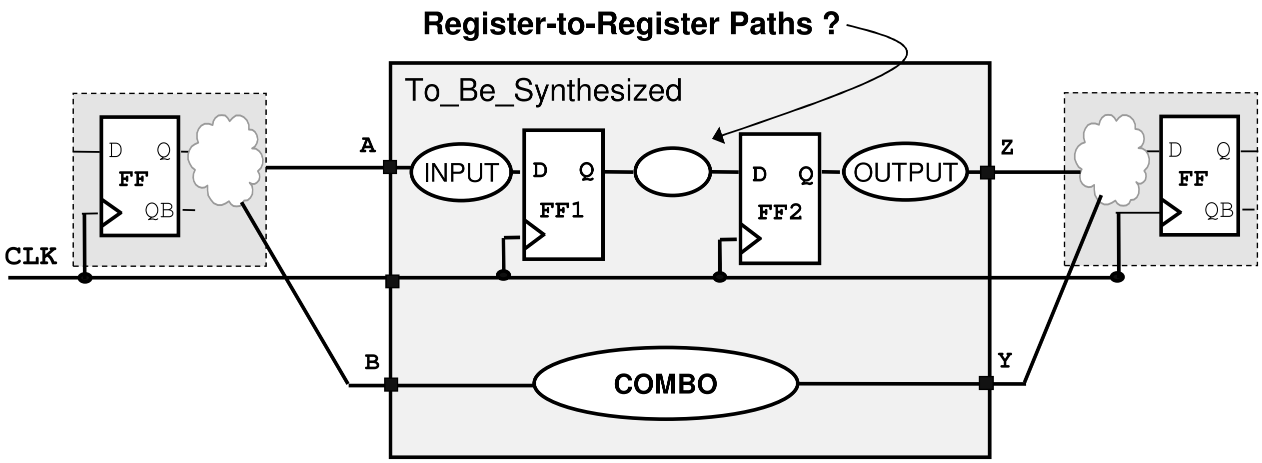
If the sub-critical paths are intertwined with the critical path (they share logic), it is possible that by improving one or more sub-critical nets this actually helps the critical path as well.

Another advantage is that you will end up with fewer violations to contend with after synthesis, an advantage if you will be using a timing-driven physical design or Placement tool: It is much easier for the physical design tool to fix a small number of violations through savvy placement, compared to having to handle a large number of violations.

Another issue is discussed on the next slide …

Problem: Reg-to-Reg Paths Ignored

Assume that, maybe due to poor partitioning, the input and output constraints are inaccurate…



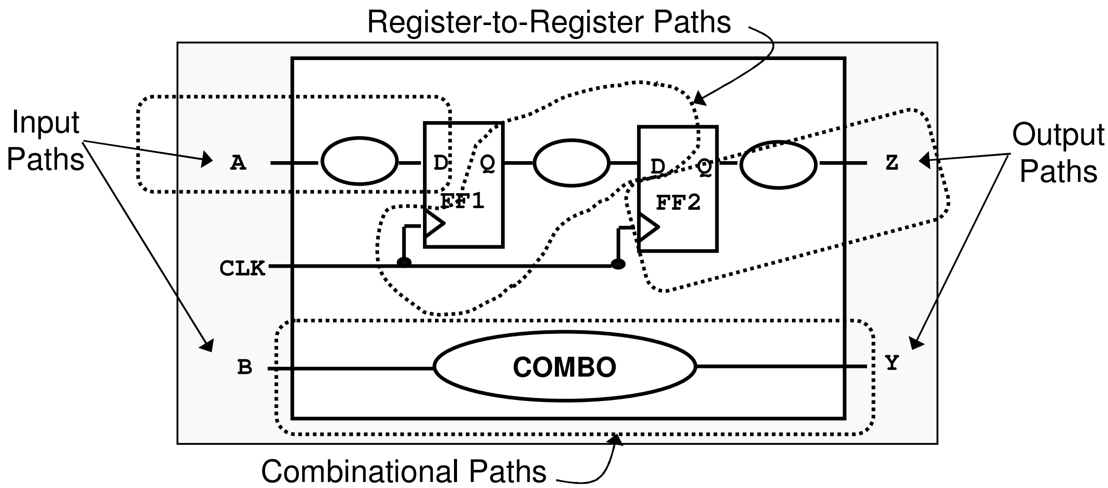
What happens if the "critical path" is an I/O path and DC gives up on it?

11-39

ANSWER: Since this is a single-clock design, all paths are in the same path group. If DC gives up on the critical path, no optimization is performed on the less critical paths, including the reg-to-reg paths. The reg-to-reg path are very accurately constrained (since they are only dependent on the clock waveform, which is usually well defined), yet they will be totally ignored due to the poor I/O constraints. This is not good.

Solution: User-Defined Path Groups

- **Custom path groups allow more control over optimization**
  - Each path group is optimized independently
  - Worst violator in one path group does not prevent optimization in another group



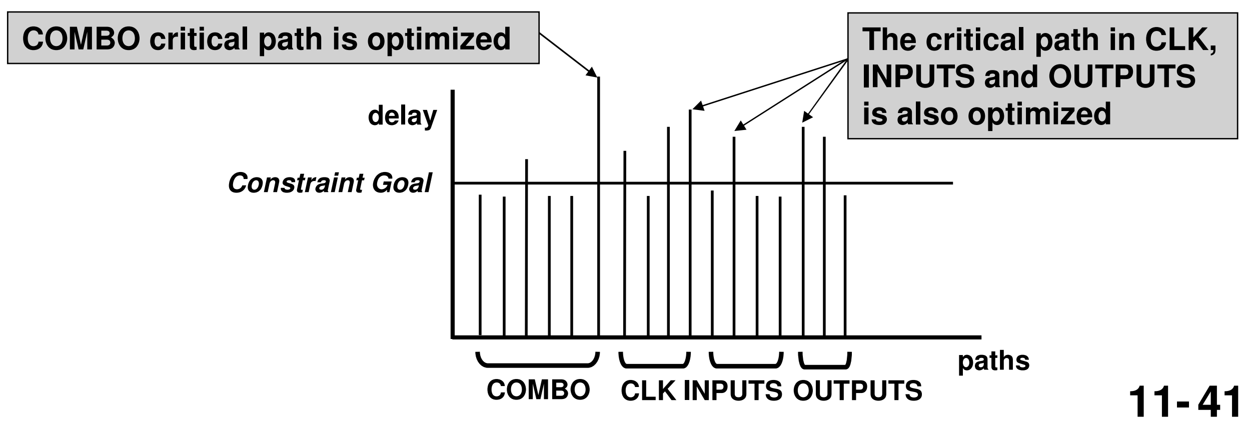
11- 40

Creating user-defined path groups can also facilitate a divide-and-conquer timing analysis strategy, since `report_timing` reports each path group's timing separately. This can help you isolate or analyze problems in a certain region of your design.

Creating User-defined Path Groups

```
# Ensure that the reg-reg paths get optimized
group_path -name INPUTS -from [all_inputs]
group_path -name OUTPUTS -to [all_outputs]
group_path -name COMBO -from [all_inputs] -to [all_outputs]
```

**Where are the reg-to-reg paths?
Are the COMBO paths in three path groups?**



11-41

The reg-to-reg paths remain in the CLK path group, which is created by default by DC during synthesis. The `group_path` command can be used for CLK to assign a non-default *weight* or *critical range* (to be discussed) to the group, but is not needed not to define the reg-to-reg paths.

A path can only be in one path group, but according to the path group arguments, the combo paths can be part of the INPUTS, OUTPUTS or COMBO group – so where are they?

The COMBO paths wind up in the COMBO group. Assuming the commands are executed in the order listed in the slide, they are first moved from CLK to INPUTS, because they match the startpoint argument `-from [all_inputs]`. They will not be moved to the OUTPUTS group, even though their endpoints match `-to[all_outputs]`, because “`-from`” has priority over “`-to`”. They finally end up in the COMBO group because their startpoints **and** endpoints match the `-from` AND `-to` arguments. DC works this way to prevent having different results if the command sequence changes!

Use `report_path_group` to get a summary of the path groups in the design.

Prioritizing Path Groups: `–weight`

- **Path groups can be given a relative priority, or *weight***
- **Allows path improvements in a given group that may degrade another group's worst violator, if the overall cost function ($\sum_{\text{neg\_slack X weight}}$) is improved**
- **Recommendations**
  - Apply a weight of 5 to the most critical paths (reg-to-reg)
  - Apply a weight of 2 to less critical paths
  - A default weight of 1 is assigned to all other paths (e.g. inaccurate I/O paths)

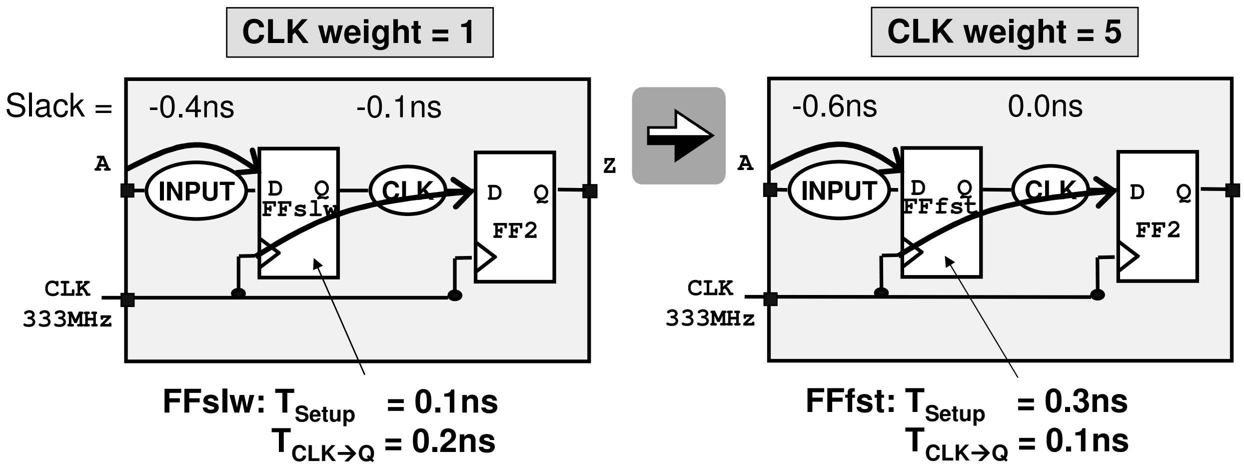
```
group_path –name INPUTS –from [all_inputs]
group_path –name OUTPUTS –to [all_outputs]
group_path –name COMBO –from [all_inputs] –to [all_outputs]
group_path –name CLK –weight 5
```

11- 42

It is possible, and recommended, to assign *weights* to different path groups by using the `–weight` option. This is a way to control the relative priority of violations. Priority is given to the path with the highest "Cost" = Negative slack X Weight. For example: A path group with a critical violation of –2ns and a *weight* of 5 has a *cost* of 10 and will therefore have higher priority than another path group with a critical violation of –3ns but a default *weight* of only 1 - *cost* of 3. If these two paths are related such that improving one hurts the other, DC will favor improving the higher weight path, even though its slack is less than the lower weight path.

Example: `–weight`

```
group_path -name INPUTS -from [all_inputs]
group_path -name OUTPUTS -to [all_outputs]
group_path -name COMBO -from [all_inputs] -to [all_outputs]
group_path -name CLK -weight 5
```



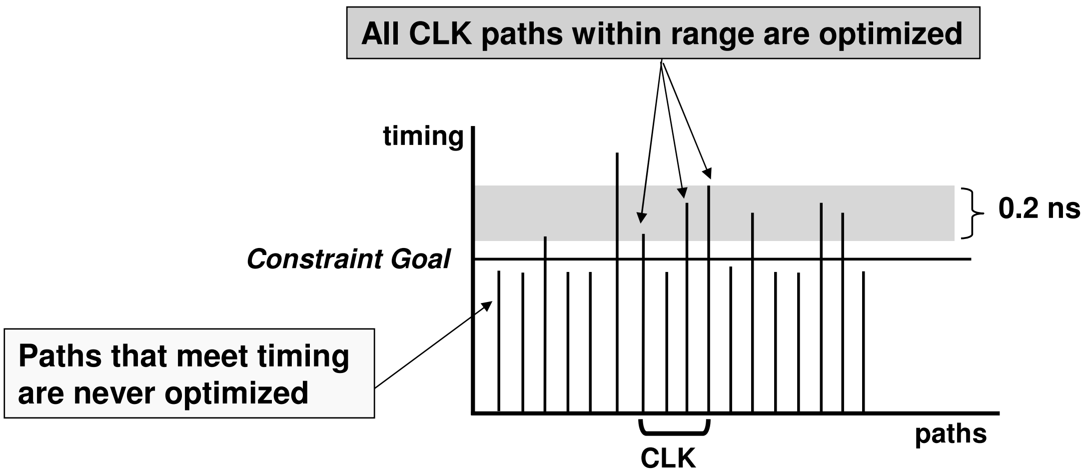
11- 43

The example above illustrates how it is possible that by adding a path group, in this case INPUTS, the worst violating path in the design, the input path with a slack of -0.4ns, actually gets worse (-0.6ns), in favor of helping the higher priority reg-to-reg path. This happened because this design change reduced the overall cost function. The cost function for the circuit on the left is [(0.4 x 1) + (0.1 x 5)] = 0.9. For the circuit on the right the cost function is 0.6. DC was able to improve the reg-to-reg timing path by switching to a register (FF1) with a faster CLK→Q delay, while giving up some setup time.

Without the `–weight` option applied to the CLK group both paths have a default weight of 1 and DC would not have considered this FF swap, which would have *increased* the overall cost from 0.5 to 0.6.

Applying a Critical Range

```
group_path -name CLK -weight 5 -critical 0.2
```



- **The critical range is with respect to the worst or critical delay**
- **Should not exceed 10% of the group's effective clock period**
- **Fixing related sub-critical paths may help the critical path**

11- 44

By default the critical range of all paths is zero.

Fixing related sub-critical paths may help the critical path. Since the critical range is with respect to the critical path delay, if the critical path delay is improved, the critical range band moves lower, along with the improved critical path.
Critical range optimization will not improve a sub-critical path if the improvements make the critical path worse.

With a critical range, DC will reduce TNS (Total Negative Slack) in the design even if it cannot reduce the WNS (Worst Negative Slack).

Complete Example

```
# Example: Assign a critical range to each path group
group_path -name CLK1 -critical_range 0.3 -weight 5
group_path -name CLK2 -critical_range 0.1 -weight 5
group_path -name CLK3 -critical_range 0.2 -weight 2
group_path -name INPUTS -from [all_inputs]
group_path -name OUTPUTS -to [all_outputs]
group_path -name COMBO -from [all_inputs] -to [all_outputs]
report_path_group
```

? What is the *weight* and the *critical range* of the INPUTS, OUTPUTS and COMBO path groups?

11-45

ANSWER: The *weight* is the default value of 1, and the *critical range* is the default value of 0ns.

In the above example the *CLK1* and *CLK2* groups are considered the most critical groups, and are therefore given a weight of 5. The *CLK3* group is still critical, but less so than *CLK1* and *CLK2*, and is therefore given a weight o 2. Each clock group is assigned a different critical range representing 10% of their respective clock periods (0.3, 0.1 and 0.2, resp.) The INPUTS, OUTPUTS and COMBO groups are not considered critical, since the input and output delay constraints are estimated on the conservative side. If the I/O paths ARE critical you may apply the appropriate weight and critical range values.

The above solution is a very CPU, and memory-intensive option, but it is worth considering. Because DC by default works only on the most critical path in each path group, it will not go very far if it gets stuck on one path that it cannot get to meet timing. By opening up the "window" to allow more paths to be optimized, DC can do a better job on the rest of the design, even at the cost of that one path. You can limit the number of non-critical paths that are optimized, and thereby keep the run time reasonable, by setting their critical range to zero.

Path Group Recommendations

Before the first compile use
`group_path -weight -critical_range`
if you want to separate I/O paths from reg-to-reg paths, or you want to focus optimization on particular paths

Recommended with DC Ultra (`compile_ultra`)

Recommended with DC Expert (`compile`)

11-46

The critical range should not exceed 10% of the clock period, or of the maximum delay constraint for the path. Apply a critical range only on "critical" path groups, e.g. reg-to-reg. The critical range should be zero (the default) on all non-critical paths.

Apply a weight of 5 to the most critical paths, a weight of 2 to the less critical paths, and the default of 1 to the rest.

Test for Understanding 1/4

1. **The DesignWare library**
   - a) Requires additional library variable settings prior to `compile_ultra`
   - b) Uses *operator inferencing* to synthesize a wide variety of arithmetic and relational operators
   - c) Uses *operator inferencing* to synthesize a wide variety of standard IP, e.g. FIFOs, shift-registers, sequential dividers
   - d) All of the above
2. **Register repositioning with `optimize_registers`**
   - a) Splits/merges registers – does not optimize combinational logic
   - b) May increase the number of register stages in the pipeline
   - c) May increase delay violations to reduce the register count
   - d) May result in faster and smaller pipelined designs

11- 47

Answers:

1. B. `compile_ultra` automatically sets the library variables; non-arithmetic/relational IP can not be *inferred* by DC – they must be *instantiated* in the RTL code
2. D. `optimize_registers`: Does not change the number of register *stages*; May increase the number of registers in a *given* stage (through splitting); Will attempt to reduce the number of registers by taking advantage of *positive* delay slack, not negative slack; Will perform an *incremental compile* at the end to optimize the new groupings of combinational logic to further reduce delay and/or area

Test for Understanding 2/4

3. **Auto-ungrouping**
   a) Is automatically invoked with `compile_ultra`
   b) Makes ‘smart’ area- or delay-based ungrouping choices
   c) Should be controlled through variable settings for best QoR
   d) All of the above
4. **By increasing the cost priority of delay DC will not fix any DRC violations – True or False?**
5. **In a single-clock design DC does not automatically create any path groups, by default – True or False?**
6. **By default optimization within a path group stops when DC ‘gets stuck’ on the critical path – True or False?**

11- 48

Answers:

3. D
4. False. DC *will* fix DRC violations as long as it can do so without creating or increasing negative delay slack.
5. False: DC creates at least one path group for the clock. If there are any unconstrained paths, these paths are grouped into a path group called *default*.
6. True. When DC get’s stuck on the critical path, by default, the sub-critical paths are not optimized. DC moves on to the next path group.

Test for Understanding 3/4

7. **Why is it recommended to optimize ignored *near-critical* paths?**

8. **Near critical paths that would be ignored by default can be optimized by**

   a) Placing them in their own *path group*

   b) Applying a *critical range* to `create_clock`

   c) Applying a *weight* to their *path group*

   d) All of the above

9. **By applying a `-weight` option to a *path group* it is possible to worsen the worst violator in another path group – True or False?**

11- 49

Answers:

7. By optimizing near critical paths: DC may be able to improve the critical path if the paths are "related"; The design ends up with fewer violating paths, which are easier for down-stream layout or physical design tools to fix; Well-constrained reg-to-reg paths may be ignored due to over-constrained I/O paths – not good.

8. A. The *critical range* is applied to *path groups*, not the clock constraint; Applying a *weight* does not direct DC to optimize otherwise-ignored paths - it only applies more "cost" weight to paths that are already being considered for optimization

9. True, if DC can reduce the overall delay cost function.

Test for Understanding 4/4

10. **Topographical mode generally gives better speed/area results than DC Ultra in WLM mode – True or False?**

11. **Including physical constraints in Topographical mode**
    a) Results in better speed/area optimization
    b) Produces a placed design which is ready for clock tree synthesis and routing
    c) Is optional, but recommended for even better correlation to post-synthesis timing
    d) All of the above

11- 50

Answers:

10. False. Topographical mode results in better timing *correlation* to the actual post-synthesis layout or physical design, not necessarily better speed and/or area results.
11. C. While topographical mode does perform an under-the-hood placement, this is only for wireload calculation purposes. The placed design can not be written out (saved) – only the netlist is saved and can be transferred to the physical design tool.

Unit Agenda

Techniques for improving synthesis results in designs with:

- Arithmetic components: *DesignWare*
- Pipelines: *Register repositioning*
- Poor hierarchical partitioning: *Auto-ungrouping*
- Aggressive DRC requirements: *Cost priority*
- Specific paths requiring more optimization focus: *Path groups*

The recommended flows

- Selecting the appropriate flow
- Detailed description of each flow
  - DC Ultra – Topographical
  - DC Ultra – WLM
  - DC Expert

11- 51

Selecting the Appropriate Flow

Preferred order ↓

Flow \ Available?	Expert License	Ultra License	DW License	Physical Libraries
DC Ultra – Topo	✓	✓	✓	✓
DC Ultra – WLM	✓	✓	✓	X
DC "Pseudo-Ultra" [1]	✓	✓	X	N/A
DC Expert	✓	X	✓ or X	N/A

✓: License or library is available; X: License or library is not available; ✓ **or** X: Applicable if available or not;
N/A: Physical libraries are not applicable to non-Ultra flows

11-52

If you have an Ultra and a DesignWare license you should always run the DC Ultra flow. You will usually get better quality of results compared to the DC Expert flow. Furthermore, if you have the physical libraries, you should run DC Ultra in "Topographical Mode" – this mode eliminates the use of WLMs and produces better timing correlation to the physical design.

[1] If you have Ultra license(s) but no DesignWare license(s), you will not be able to run the Ultra flow. You can however enable a sub-set of the Ultra optimizations within the Expert flow by setting certain attributes and variables. These "pseudo-Ultra" flows will not be discussed but are provided in the appendix for your reference.

If you do not have an Ultra license you will only be able to execute the Expert flow. In a design with arithmetic or relational operators (+, -, *, >, <, =) you will generally achieve better results if you have a DesignWare license, but this is not required for the Expert flow.

DC Expert Flow - No Ultra License

- Enable DW library if license available
- Create path groups if I/O constraints are not accurate

```
compile -b -scan -map high
```

```
compile -b -scan -map high -incr (-area ..)
```

- Increase max-delay cost priority if acceptable
- Apply more focus on violating critical paths
- Re-partition if sub-designs are poorly partitioned

```
compile -b -scan -map high -incr (-area ..)
```

Note for all flows: Check constraints after each optimization step. Continue only if still violating.

11- 53

DC Expert Flow - No Ultra License

compile_expert.tcl

```
        # Enable DW library if available
set synthetic_library dw_foundation.sldb
lappend link_library $synthetic_library
        # Create path groups for I/O/combo paths if I/O constraints are not accurate
group_path –critical range <10% of max delay> –weight 5|2|1 [1]

compile –boundary –scan –map_effort high
        # Continue if NOT meeting constraints
compile –boundary –scan –map_effort high –incremental \
                                (–area_effort[2] medium|low|none)
        # Continue if NOT meeting constraints:
        # Make max-delay higher priority if acceptable to postpone DRC fixing
set_cost_priority –delay
        # Apply more focus on violating critical paths
group_path –critical range <10% of max delay> –weight 5|2|1
        # Repartition the design by adding the –ungroup_all
        # option to the compile below, or with
group | ungroup
compile –boundary –scan –map_effort high –incremental \
                                (–area_effort medium|low|none)
```

11- 54

[1] Recommendations for `group_path`:

A) Before the two-pass `compile+compile –inc` or `compile_ultra`:

1) If I/O constraints are known to be estimates or inaccurate:
   - Separate input, output and combo logic path groups from reg-to-reg path groups
   - Apply a critical range of 10% of clock period (P) or of the maximum path delay constraint to the clock and any other accurately constrained groups, plus a weight of 5 for the most critical, and 2 for the less critical path groups. None for IOs.
2) If you know of specific paths that are more important or more timing critical than others:
   - Separate these paths from the others by creating a separate path group
   - Apply a critical range of (0.1 x Max delay) and apply a weight factor of 2-5

B) Before the next incremental compile, if you still have timing violations, analyze the violations:
If you discover that additional "important" paths are violating, while less important paths within
the same group are meeting timing or violating by less:
- Separate these paths from the others by creating a separate path group
- Apply a critical range of (0.10 x Max delay) and apply a weight of 5

[2] The -`area_effort` option is automatically set to the same value as `–map_effort`. In a timing-critical design, if you can afford to give up area to possibly reduce run time, set the –`area_effort` to something less than "high", e.g. *medium*, *low*, or *none*.

DC Ultra Flow - WLM Mode

- Apply auto-ungrouping settings
- Create path groups if I/O constraints are not accurate
- Relax constraints of pipelined sub-designs

↓

```
compile_ultra -scan -retime -timing|-area
```

↓

- Reset constraints of pipelined sub-designs
- Apply pipeline attribute to pipelined sub-designs

↓

```
optimize_registers -only_attributed_designs
```

↓

- Apply more focus on violating critical paths
- Enable Ultra optimization for incremental compile

↓

```
compile_ultra -scan -incr
```

11- 55

DC Ultra Flow - WLM Mode

compile_ultra_wlm.tcl

```
        # Apply practical auto-ungrouping settings 1
set compile_auto_ungroup_delay_num_cells <max_limit>
set compile_auto_ungroup_count_leaf_cells true
set compile_auto_ungroup_override_wlm true
set_ungroup <top_level_and/or_pipelined_blocks> false
        # Create path groups for I/O/combo paths if I/O constraints are not accurate
group_path -critical range <10% of max delay> -weight 5|2|1
        # If design contains pipelined sub-designs and the pipeline registers
        # are grouped together at the input or output (recommended)
set_multicycle_path -setup <#stages>2 -from|-to <pipeline_flops>
        # First compile
compile_ultra -scan -retime -timing|-area 3
reset_path -from|-to <pipeline_flops>
        # Continue if pipeline violates timing; Skip if no pipeline issues:
set_optimize_registers true -design <pipelined_designs>
optimize_registers -only_attributed_designs
        # Continue if design is NOT meeting all constraints:
        # Apply more focus on violating critical paths
group_path -critical range <10% of max delay> -weight 5|2|1
compile_ultra -scan -incremental 4
```

11-56

[1] `compile_ultra` (even with `–area`) performs delay-based auto-ungrouping, therefore the `compile_auto_ungroup_area_num_cells` variable is not set here.

[2] The `–setup` value, shown as `#stages`, corresponds to the number of stages that the logic will be split into: If the pipeline input or output remains registered, the number equals the pipeline latency or the number of registers in the design; If logic is allowed on both the input and output, the number equals latency + 1.

[3] `compile_ultra` unconditionally ungroups all hierarchical DesignWare components (except special 'built-in pipeline' parts) . If you wish to disable this use: `set compile_ultra_ungroup_dw false` (`true` by default).
See SolvNet article **Doc Id:** 017448 "Encapsulation" (for details on *compile_ultra –timing/-area*).

[4] The `–incremental` option for `compile_ultra` is available in WLM mode (non-topographical mode) starting with DC v2007.03. If using an earlier DC version execute the following instead:

```
set_ultra_optimization true
compile -boundary -scan -map high -incremental \
                                   (-area medium|low|none)
```

DC Ultra Flow - Topographical Mode

- Invoke DC in Topographical mode

- Apply auto-ungrouping settings
- Create path groups if I/O constraints are not accurate
- Relax constraints of pipelined sub-designs
- Apply or extract physical constraints (if available)

```
compile_ultra -scan -retime -timing|-area
```

- Reset constraints of pipelined sub-designs
- Apply pipeline attribute to pipelined sub-designs

```
optimize_registers -only_attributed_designs
```

- Apply more focus on violating critical paths

```
compile_ultra -scan -incr
```

- Write out physical constraints

11- 57

DC Ultra Flow - Topographical Mode

UNIX% `dc_shell-t -topographical #Invoke DC in Topographical Mode`

compile_ultra_topo.tcl

```
        # Same as DC Ultra – WLM Mode:
        # - Set auto-ungrouping variables
        # - Create path groups for I/O/combo paths if I/O constraints are not accurate
        # - Apply multi-cycle paths to arithmetic pipelines

        # If the floorplan is available, apply or extract the physical constraints
source <physical_constraints_file> OR
extract_physical_constraints <DEF_file>

        # First compile
compile_ultra -scan -retime -timing|-area
reset_path -from|-to <pipeline_flops>
        # Continue if pipeline violates timing; Skip if no pipeline issues:
set_optimize_registers true -design <pipelined_designs>
optimize_registers -only_attributed_designs
        # Continue if design is NOT meeting all constraints:
        # Apply more focus on violating critical paths
group_path -critical range <10% of max delay> -weight 5|2|1
compile_ultra -scan -incremental[1]
write_physical_constraints -output PhysConstr.tcl[2]
```

See Appendix 3
for more details

11-58

[1] The `-incremental` option for `compile_ultra` in Topographical mode is available starting with DC v2006.06.

[2] Starting with DC v2007.03 the physical constraints are saved in *ddc*, so it is no longer required to explicitly write out the physical constraints to import into IC Compiler.

Summary: Unit Objectives

You should now be able to:

- **Describe five additional DC techniques for improving synthesis results - decide when they are applicable**
- **Select and execute the most appropriate *compile flow***
  - DC Ultra - Topographical
  - DC Ultra – WLM
  - DC Expert

11- 59

Lab 11: Synthesis Techniques

90 minutes

Select an appropriate compile flow.

Create and verify a complete *compile* run script.

Synthesize a design using the appropriate synthesis techniques.

Analyze the results after each key step.

11- 60

Appendix 1

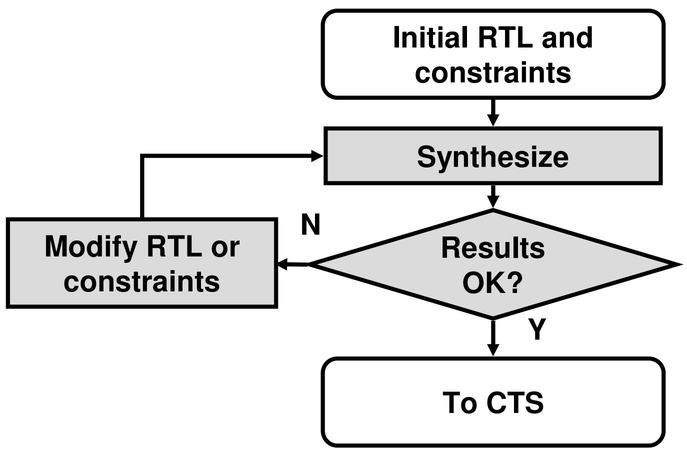
Parallel Synthesis using ACS – Automated Chip Synthesis

The “Pseudo” Ultra flow can be used if you have Ultra license(s) but do not have DesignWare license(s).

Need for Faster Compile Times

You may be doing design exploration - your design methodology therefore requires that you perform many synthesis iterations in a relatively short time, and your design is very large

How can you dramatically reduce compile times without impacting QoR?



11- 62

Answer: Perform parallel synthesis with “Automatic Chip Synthesis” or ACS.

Automated Chip Synthesis

ACS -- Automated Chip Synthesis
"Parallel synthesis, the easy way"

A single compile command:

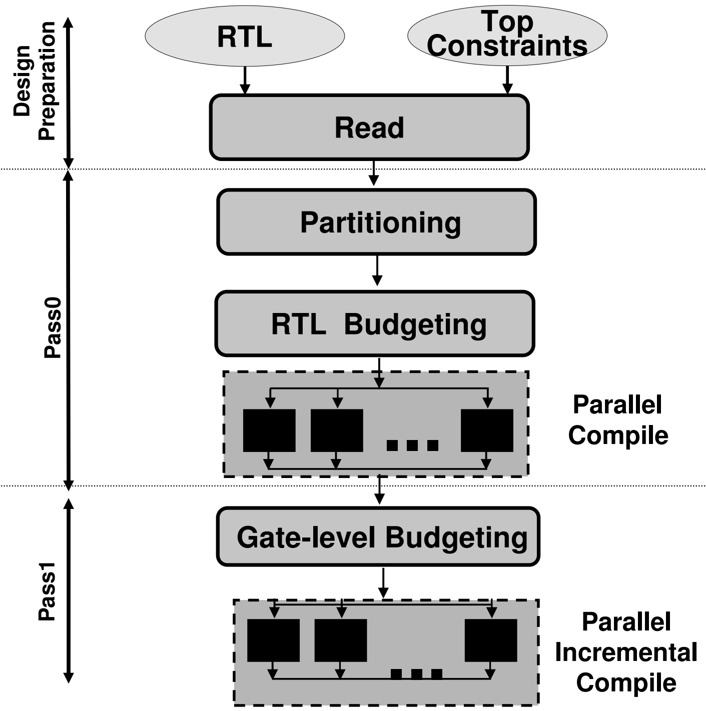
- **Automatically divides the design into manageable sub-designs or "compile partitions"**
- **Creates budgets and compile scripts for each sub-design**
- **Distributes the sub-designs onto multiple CPUs and performs parallel block-level synthesis**
- **Pulls all compiled blocks back together**
- **Generates all the necessary reports to analyze QoR**

11- 63

ACS is really more than a single command. But once you have the environment properly set up, one single command distributes and executes all the block-synthesis jobs and then ties all the results back together. If the results don't meet your constraints, another single command sets up, distributes, and executes a recompile strategy, and again ties the results back together.

Obviously a design can be turned around quicker if many blocks can be compiled in parallel, as compared to a single top-down compile on the whole design. Although in general top-down compiles produce better QoR (Quality of Results), the efficient budgets and multi-pass compile strategies used by ACS typically produce comparable results in much less time. The parallel compile jobs can be submitted to multiple CPU's explicitly, or passed to the appropriate CPU if using common load sharing tools.

Basic ACS Flow and Commands




```
read_verilog|vhdl
source top_const.tcl
```

```
acs_compile_design
```

```
acs_refine_design
acs_recompile_design
```

11-64

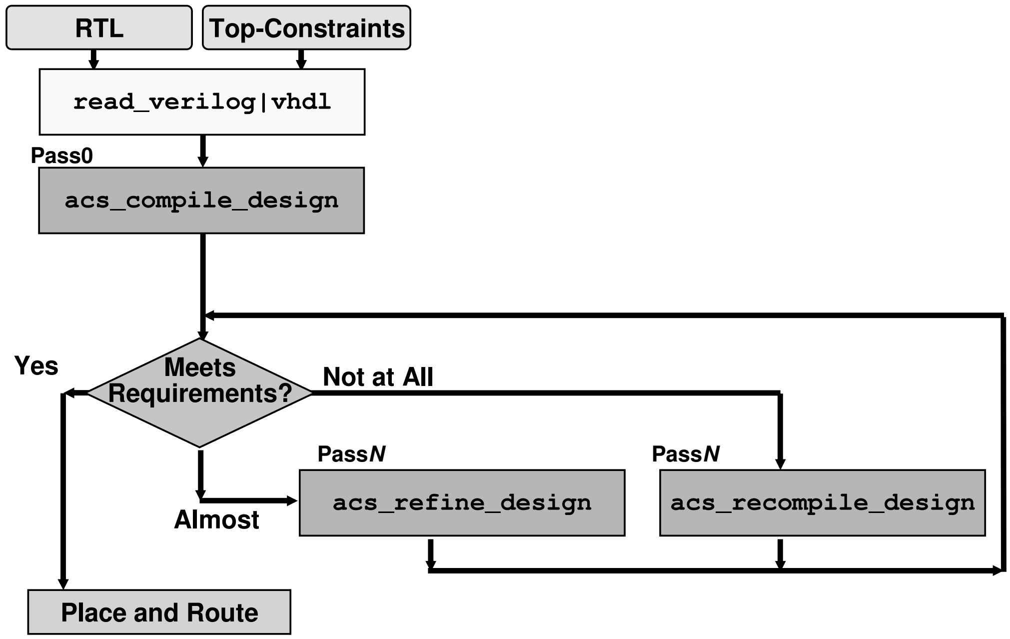
The commands on the right demonstrate the ease-of-use aspect of ACS. These are all the ACS commands you need to run a two-pass compile. The set-up commands and variables are not shown here.

Automated Chip Synthesis uses the following designs by default as the compile partitions:

- First-level sub-designs (hierarchical children of the top-level design)
- Reference designs of multiple instances

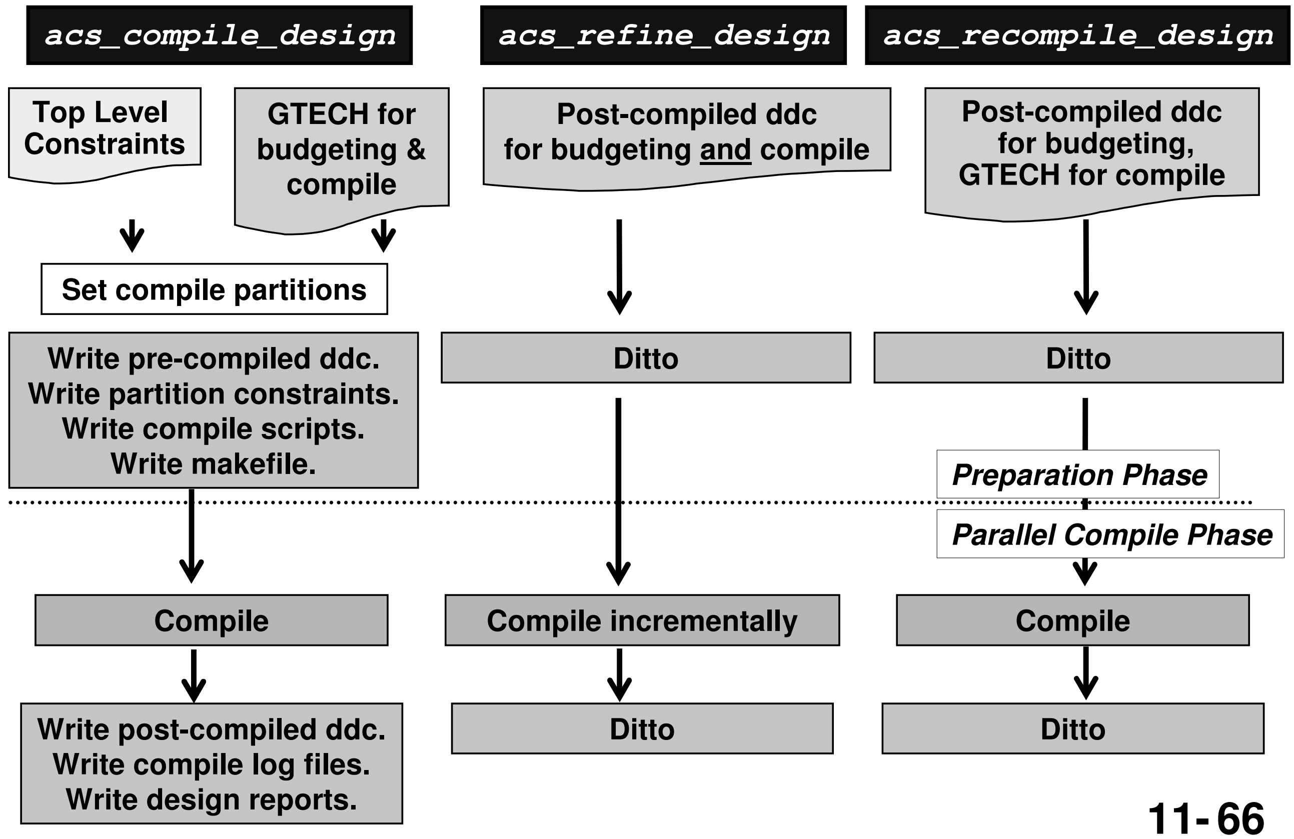
The user can modify this default partitioning behavior, as needed.

ACS Flow is Flexible



11- 65

Comparing the Different ACS Compiles

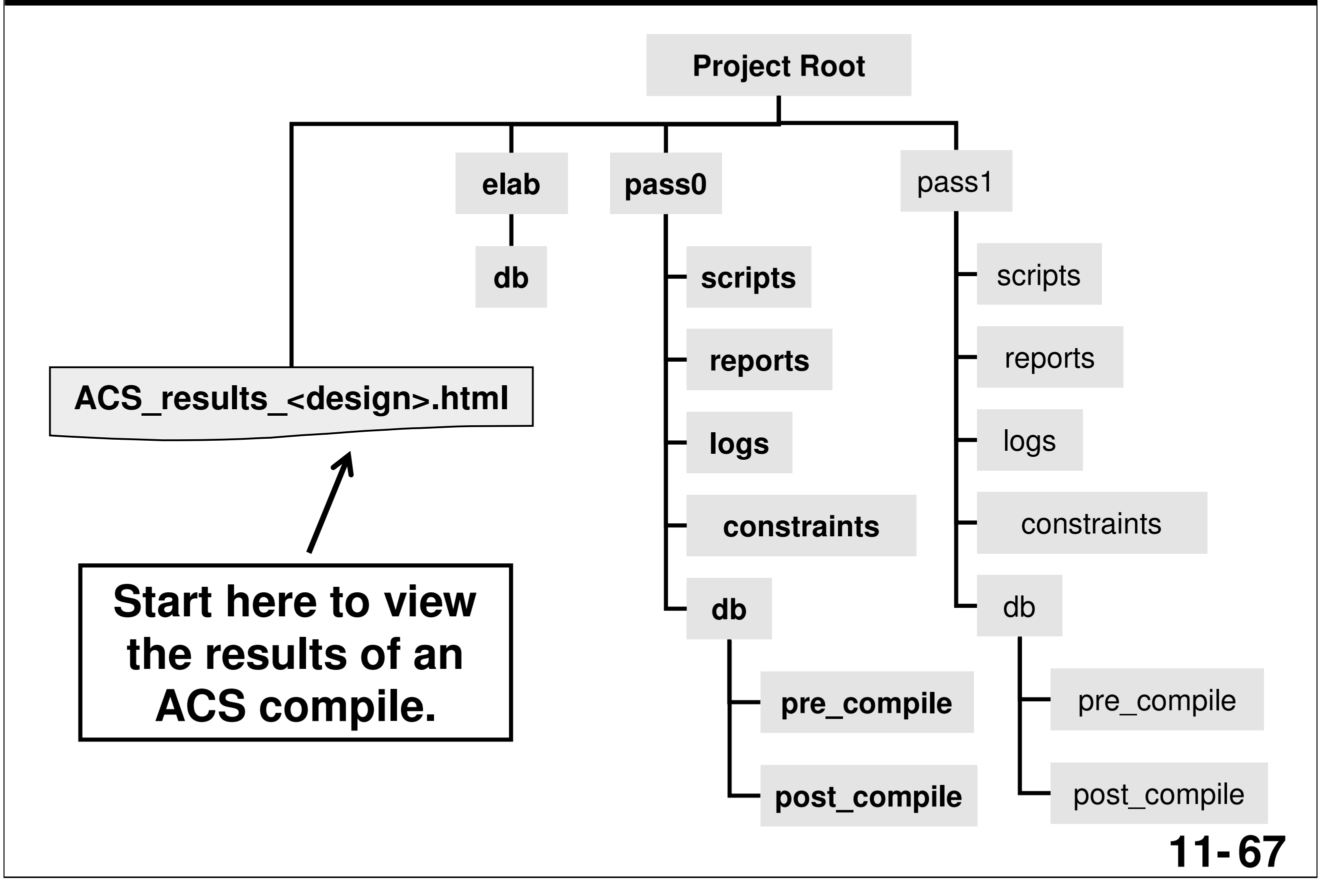


11-66

The commands `acs_refine_design` and `acs_recompile_design` will run with same, fine-tuned 2nd pass constraints file, but on a different representation of the design:

The `acs_recompile_design` command re-compiles the un-mapped, or GTECH representations of the original design, while `acs_refine_design` performs incremental compiles on the resulting netlists from pass-0. The *recompile* command starts from an un-mapped or "unbiased" start-point, but with much more accurate constraints. This gives DC more flexibility to meet the constraints during the architectural-, logic- and gate-level optimization phases. The *refine* command runs faster but limits the optimization to *incremental mapping*, suitable for small remaining violations.

ACS Directory Structure



ACS automatically generates the above directory structure, which contains the scripts, designs, constraints, logs and reports. The user can customize the directory structure and all file names.

All the reports, logs, constraints and scripts can be conveniently accessed through an *html* page, which is also automatically generated at the end of the compile.

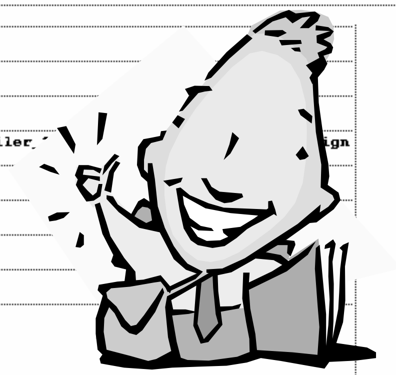
The Unix directory `elab` is created to support “update mode”. This allows small changes to be made to the design - ACS then recompiles only the affected compile partitions, not the entire design. The `elab` Unix directory is populated only if `acs_read_hdl` is used to read in the design, and is executed in *update* mode using an optional switch `-auto_update`.

View the Results of a Compile

Navigate the reports, log files, scripts and constraints from a compile using your web browser!

ACS HTML report for design RISC_CORE

Design	RISC_CORE
Destination	pass0
Date	Sat Jan 29 23:20:21 2005
Work Directory	/remote/training/projects/DesignCompiler_ ... ign
No. of Partitions	9
User Defined Compile Script	
User Defined Constraint	
Environment File	env
Reports	report_area report_timing report_constraints report_qor



Partition Name	Script(?)	Constraint(?)	Log File	Errors	Warnings
RISC_CORE (top design)	partition run script	budgeted constraint	log file	0	0
ALU	partition run script	budgeted constraint	log file	0	0
CONTROL	partition run script	budgeted constraint	log file	0	0

11- 68

Analyzing an Unsuccessful Run

If the compile run does not finish successfully (aborted, error, etc), an HTML summary file is not generated. In this case, you can generate the HTML file by running the `acs_write_html` command and then use the information to analyze why the compile run failed.

Examples of Areas You Can Customize

- **The directory structure and file naming conventions**
- **The following steps in the default flow**
  - Generating the makefile
  - Resolving multiple instances
  - Identifying compile partitions
  - Generating partition constraints
  - Generating compile scripts
  - Running the compile job (i.e. how the compile job is invoked and which executable file is used)
- **The default behavior of the ACS compile commands**

```
help acs*

printvar acs*
```

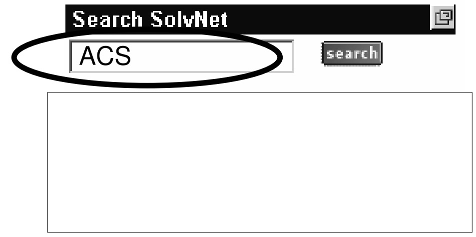
11- 69

The default ACS flow successfully optimizes most designs.

However in those cases where the default flow is not sufficient, ACS provides the capability to customize.

For More Information on SolvNet

Search the *documentation* or *articles* using the key word “ACS”



Documentation examples:

- **Automated Chip Synthesis Tutorial (Version Y-2006.06)**
- **Variables (Version Y-2006.06)**
- **Customizing Automated Chip Synthesis (Version Y-2006.06)**

Article example:

- **“Improving Automated Chip Synthesis Results”**
  Product: Design Compiler Doc ID: 010246 Last Modified: 03/05/2004

11- 70

ACS Recommendations

If your compile times are too long, and you have multiple DC licenses and CPUs, use ACS to perform automated *parallel synthesis*

If multiple Ultra/DW licenses available: ACS “Ultra”

If multiple Expert licenses available: ACS “Expert”

11- 71

Selecting the Appropriate Flow

Flow \ Available?	Expert License	Ultra License	DW License	Physical Libraries
DC Ultra – Topo	✓	✓	✓	✓
DC Ultra – WLM	✓	✓	✓	X
DC "Pseudo-Ultra"	✓	✓	X	N/A
DC Expert	✓	X	✓or X	N/A
ACS Expert or Ultra	✓✓✓	✓✓✓ or X	✓✓✓ or X	N/A

Licenses for ACS

✓: License or library is available; X: License or library is not available; ✓ **or** X: Applicable if available or not;
✓✓✓: Multiple licenses available; **N/A**: Physical libraries are not applicable to non-Ultra flows

11-72

If you have a very large design for which the compile time would be prohibitively long, AND you have ***n + 1*** licenses, you can take advantage of the ACS flow to perform "distributed processing" and significantly reduce the overall compile time. With ACS the entire design is read in on the "master" machine with enough memory (possibly a 64-bit machine). ACS then automatically partitions the design into ***n*** number of smaller sub-designs and derives budgets or constraints for each of these sub-designs based on the provided top-level constraints. These constrained sub-designs are then farmed out to ***n*** machines (which do not require as much memory), where they are compiled in parallel, and are then integrated back together again on the "master" machine. ACS can perform Ultra optimizations if the appropriate number of DC Ultra and DesignWare licenses are available. There is no "Topographical Mode" for ACS.

ACS Expert or Ultra Flow

- Re-partition if sub-designs are poorly partitioned
- Create path groups if I/O constraints are not accurate
- Enable Ultra optimization if licenses available
- Select compile partitions for parallel compiles
- Enable test-ready synthesis
- Enable high-effort compile

```
acs_compile_design -force
```

```
acs_recompile_design
```

OR

```
acs_refine_design
```

- Re-partition sub-designs as needed
- Increase max-delay cost priority if acceptable
- Apply more focus on violating critical paths

- Uniquify multiply-instantiated designs for layout

11- 73

ACS Expert or Ultra Flow (1 of 2)

acs_1.tcl

```
        # Ensure efficient partitioning for synthesis 1
        # Avoid uniquify-ing multiply-instantiated designs if possible 2
group | ungroup <poorly_partitioned_sub_designs>

        # Create path groups for I/O/combo paths if I/O constraints are not accurate
group_path -critical range <10% of max delay> -weight 5|2|1

        # Enable Ultra if multiple Ultra + DW  licenses are available: 3
acs_set_attribute  UltraOptimization true
acs_set_attribute  CompileUltra true

set_compile_partitions -auto|-designs|-level  -force  4
acs_set_attribute TestReadyCompile true
acs_set_attribute FullCompile high
acs_compile_design -force

        # Continue if NOT meeting timing
...
...
```

11-74

Benefit of ACS: Run-time improvement for large designs.
ACS does not currently support *topographical mode*.

[1] ACS does not perform auto-ungrouping, even with Ultra, so you must ensure that you have efficient partitioning before compiling by “manually” `grouping/ungrouping`.

[2] ACS does not perform “auto-uniquification”. If the different instances of the same design have very different “surroundings” (different loads, drivers, input and/or output delay requirements, etc) then go ahead and manually `uniquify` prior to running ACS compile. If their surroundings are very similar then do not `uniquify`. *Uniquified* designs are treated as separate partitions and will therefore increase compile time, which goes against the main advantage of ACS. However, once the ACS flow is completed you should `uniquify` the design just prior to moving on to physical design (layout).

[3] The *UltraOptimization* attribute is used by `acs_refine/recompile_design`. The *CompileUltra* attribute enables the full “`compile_ultra`” capability during `acs_compile_design`.

[4] The `-force` option forces ACS to compile even though it may have multiply-instantiated designs. By default the compile would abort.
While boundary optimization is available in ACS, it is NOT recommended, therefore the related attributes are not set here.

ACS Expert or Ultra Flow (2 of 2)

acs_2.tcl

```
        # Continue after acs_1.tcl, if NOT meeting timing

acs_recompile_design;  # For significant timing violations
acs_refine_design;     # For small timing violations

        # Continue if NOT meeting timing
        # Change partitioning, DRC priority and/or path group focus:
group | ungroup <sub_designs_or_DesignWare_parts>
set_cost_priority -delay
group_path -critical range <10% of max delay> -weight 5|2|1

        # Iterate as needed

        # Uniquify multiple instantiations in preparation for physical design
uniquify <multiply_instantiated_designs>
```

11- 75

Appendix 2

‘Pseudo’ Ultra Flows

The “Pseudo” Ultra flow can be used if you have Ultra license(s) but do not have DesignWare license(s).

DC ‘Pseudo-Ultra’ Flow - No DW license (1/2)

dc_pseudo_ultra1.tcl

```
        # Apply practical auto-ungrouping variable values 1
set compile_auto_ungroup_delay_num_cells <max_limit>
set compile_auto_ungroup_area_num_cells <max_limit>
set compile_auto_ungroup_count_leaf_cells true
set compile_auto_ungroup_override_wlm true
set_ungroup <top_level_and/or_pipelined_blocks> false
        # Create path groups for I/O/combo paths if I/O constraints are not accurate
group_path -critical range <10% of max delay> -weight 5|2|1
        # If design contains arithmetic pipelined sub-designs and the pipeline
        # registers are grouped together at the input or output (recommended)
set_multicycle_path -setup <#stages> -from|-to <pipeline_flops>

        # Enable Ultra optimizations available for compile 2
set_ultra_optimization -no_auto_dwlib true
set hlo_disable_datapath_optimization true
set compile_slack_driven_buffering true
compile -boundary -scan -map_effort high \
      -auto_ungroup delay|area (-area_effort medium|low|none)
reset_path -from|-to <pipeline_flops>
        # Continue if NOT meeting constraints ……
```

11-77

[1] `compile -auto_ungroup delay|`**`area`** uses the `compile_auto_ungroup_delay|`**`area`**`_num_cells` variable to determine the maximum block size limit for ungrouping. If doing area-based ungrouping set the `compile_auto_ungroup_area_num_cells` variable, otherwise set `compile_auto_ungroup_delay_num_cells`.

[2] If you have a DC Ultra license but no DesignWare license you can not run `compile_ultra`, but you can still get SOME Ultra benefits with the above attributes/variables/command options.

DC ‘Pseudo-Ultra’ Flow - No DW license (2/2)

dc_pseudo_ultra2.tcl

```
        # Continued from dc_pseudo_ultra1.tcl
        # Continue if pipeline violates timing; Skip if no pipeline issues:
set_optimize_registers true -design <pipelined_designs>
optimize_registers -only_attributed_designs

        # Second compile
compile -boundary -scan -map_effort high -incremental \
                              (-area_effort medium|low|none)

        # Continue if NOT meeting constraints

        # Make max-delay higher priority than DRCs
set_cost_priority -delay
        # Ungroup DesignWare components amid combinational logic 1
ungroup <DesignWare_components>
        # Apply more focus on violating critical paths
group_path -critical range <10% of max delay> -weight 5|2|1

compile -boundary -scan -map_effort high -incremental \
                              (-area_effort medium|low|none)
```

11-78

[1] Unlike `compile_ultra` which unconditionally ungroups all DesignWare components (except special ‘pipelined’ parts), `compile -auto_ungrouping` does not. If there are any DesignWare components which have combinational logic in front of, or after them, you should manually ungroup them to achieve a faster/smaller design.

ACS 'Pseudo-Ultra' Flow (1 of 2)

acs_pseudo_ultra1.tcl

```
        # Ensure efficient partitioning for synthesis
        # Avoid uniquify-ing multiply-instantiated designs if possible
group | ungroup <poorly_partitioned_sub_designs>

        # Create path groups for I/O/combo paths if I/O constraints are not accurate
group_path -critical range <10% of max delay> -weight 5|2|1

        # If multiple Ultra licenses, but no DW licenses are available
set hlo_disable_datapath_optimization true
acs_set_attribute  UltraOptimization true

set_compile_partitions -auto|-designs|-level  -force
acs_set_attribute TestReadyCompile true
acs_set_attribute FullCompile high
acs_compile_design -force

        # Continue if NOT meeting timing
...
...
```

11-79

ACS 'Pseudo-Ultra' Flow (2 of 2)

acs_pseudo_ultra2.tcl

```
        # Continue after acs_pseudo_ultra1.tcl, if NOT meeting timing

acs_recompile_design;  # For significant timing violations
acs_refine_design;     # For small timing violations

        # Continue if NOT meeting timing
        # Change partitioning, DRC priority and/or path group focus:
group | ungroup <sub_designs_or_DesignWare_parts>
set_cost_priority -delay
group_path -critical range <10% of max delay> -weight 5|2|1

        # Iterate as needed

        # Uniquify multiple instantiations in preparation for physical design
uniquify <multiply_instantiated_designs>
```

11-80

Appendix 3

Topographical mode commands and variables

(from Unit 7)

Specifying Physical Libraries and Floorplans

run.tcl

```
# Specify physical libraries for Topographical mode
create_mw_lib
         -technology           <technology_file> \
         -mw_reference_library <mw_reference_libraries> \
                               <mw_design_library_name>
open_mw_lib                    <mw_design_library_name>
set_tlu_plus_files \
                -max_tluplus   <max_tluplus_file> \
                -tech2itf_map  <mapping_file>
```

floorplan.con

```
# Physical constraints which define the floorplan for Topographical mode
set_aspect_ratio               set_port_side
set_utilization                set_port_location
set_placement_area             set_cell_location
set_rectilinear_outline        create_placement_keepout
```

11- 82

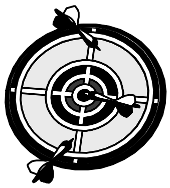
Agenda

DAY 3

9	More Constraint Considerations (Lab cont'd)
10	Multiple Clock/Cycle Designs
11	Synthesis Techniques and Flows
12	Post-Synthesis Output Data
13	Conclusion

Synopsys 10-I-011-SSG-013 12- 1

Objectives

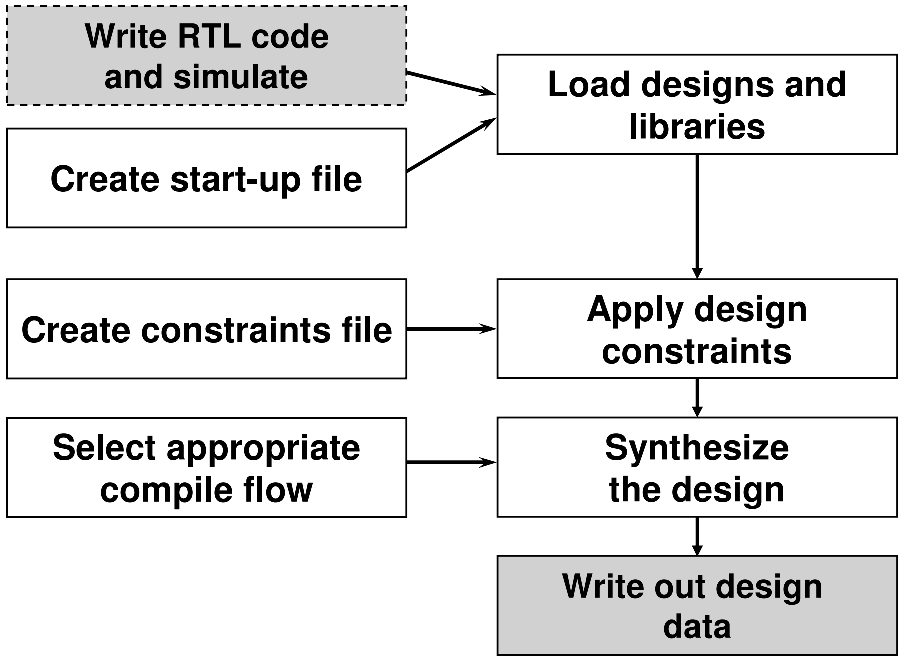


After completing this section, you should be able to:

- **Generate required post-synthesis output data files that take into account the abilities and restrictions of downstream physical design tools**
- **Select the appropriate workshop to learn more about physical design tool considerations**

12- 2

RTL Synthesis Flow



12- 3

Ideal Network Commands to be Covered

```
create_clock -period 2.5 [get_ports clk]
set_clock_uncertainty -setup 0.3 [get_clocks clk]
set_clock_transition 0.2 [get_clocks clk]
...
# Disable timing/DRC optimization of HFN Port sources
set_ideal_network [get_ports reset* select*]

# Disable timing/DRC optimization of HFN Pin sources if
# GTECH pin names and/or net names are known
set_ideal_network [get_pins FF_SET_reg/Q]
set_ideal_network -no_propagate [get_nets CTRL]

# Optional replacement values for default zero delay and
  transition values of ideal networks
set_ideal_latency 1.4 [get_ports reset* select*]
set_ideal_transition 0.5 [get_pins FF_SET_reg/Q]
```

12- 4

For scan-enable can't use `set_ideal_network` because this will prevent `insert_dft` from hooking up the scan-enable signals.

Output Data Commands to be Covered

```
...
# Insert buffers for all multiple-port nets – eliminate assign
set_fix_multiple_port_nets -all -buffer_constants
compile ...  or compile_ultra ...
# Convert tri to wire - eliminate assign
set verilogout_no_tri true
# Eliminate special characters in the netlist and constraints file
change_names -rules verilog -hierarchy
write -f ddc -hierarchy -output my_ddc.ddc
write -f verilog -hierarchy -output my_verilog.v
# Write out the constraints-only sdc file
write_sdc my_design.sdc
# Write out the scan chain information
write_scan_def -out my_design.def
# Write out the physical constraints
write_physical_constraints -output PhysConstr.tcl
```

12- 5

Note: `set_fix_multiple_port_nets` must be applied before compiling the design, because the fix happens during *compile* – it is recommended to do so before the first *compile*.

The `set verilogout_no_tri true` and `change_names` commands should be applied after all optimizations, just prior to writing out the netlist and/or constraints.

Unit Agenda

High-fanout net buffering

Data needed for physical design

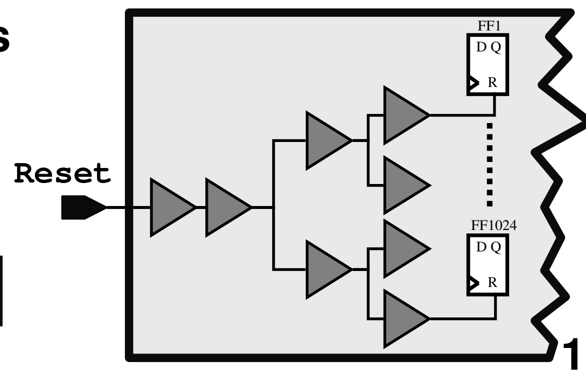
Netlist restrictions

- `assign` statement
- Special characters

12- 6

Default Optimization/Buffering Behavior

- **By default DC optimizes and buffers <u>all</u> time-constrained paths for delay and DRCs, except for clock networks, which are treated as "ideal"**
- **This includes potential "high fan-out" (HFN) signals such as**
  - Set or Reset
  - Select or Enable
- **These HFN nets can fan out to 100's or 1000's of cells requiring many buffers**



So what's the problem?

12-7

DRCs = Design Rule Constraints, which include *max_transition*, *max_capacitance* and/or *max_fanout* rules.

Clock network buffering is deferred to "Clock Tree Synthesis" (CTS), which is performed during physical design or layout. The main goal for CTS is to minimize the clock skew, in addition to meeting DRC rules. Minimizing skew is much more sensitive to parasitic interconnect differences, which is why it is typically deferred to the physical design phase. DC automatically treats clock networks (ports or pins defined as clocks, along with their entire transitive fanout) as "ideal", which means that they have zero transition times, zero insertion delay, and zero skew, so no buffering or optimization is performed.

HFN Buffering Consideration

- ***WLMs* are especially inaccurate for very large fanouts**
  - Not applicable if using *topographical* mode
- **Buffering and optimization of HFNs may therefore not be worthwhile**

The paths may end up being severely over- or under-buffered

- **Better to disable HFN buffering and defer to the physical design or layout phase**

Disabling buffering and timing optimization of these high-fanout nets is accomplished by defining them as *ideal networks*

12- 8

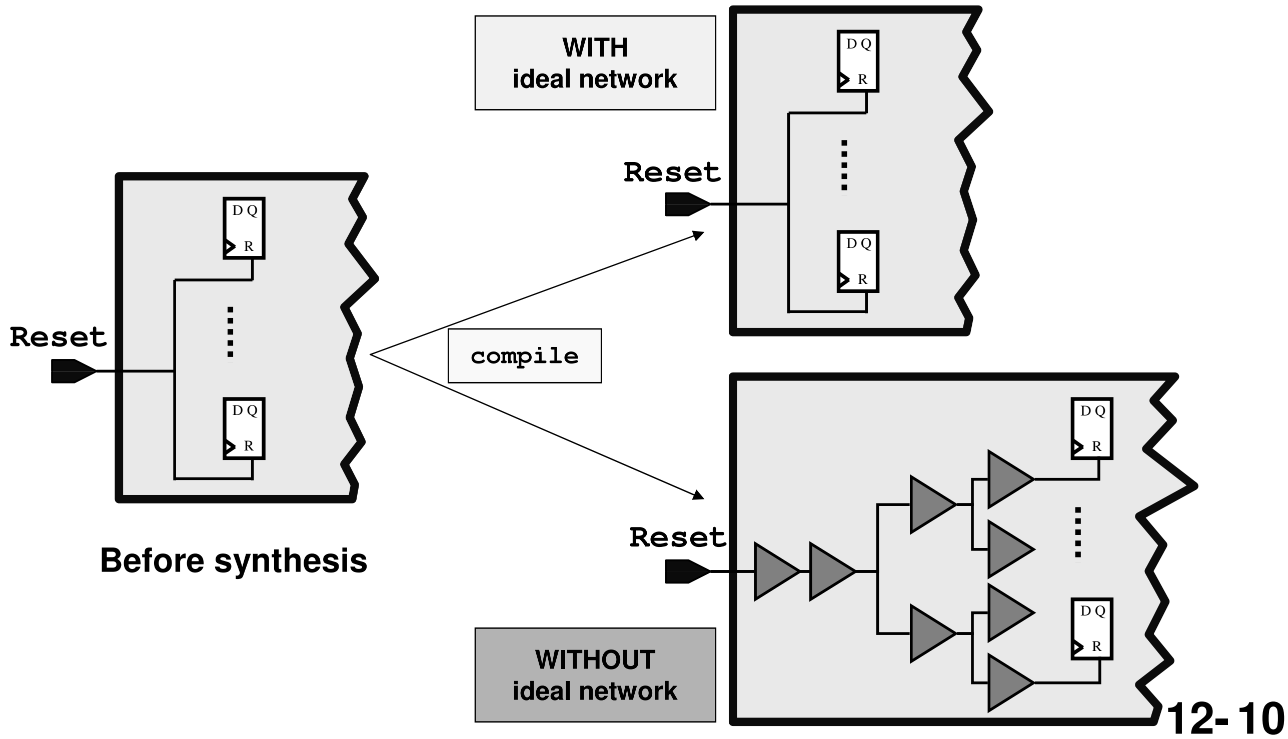
What are *Ideal Networks*?

- ***Ideal networks* have ideal characteristics**
  - Zero net and cell delays
  - Zero output transition
  - Zero pin and net capacitance
  - Zero net resistance
- **Maximum delay constraints and design rule constraints are therefore always met**
- **All nets and cells are automatically marked as *dont_touch***
- **As a result, *ideal networks* are not optimized or buffered during *compile***

12- 9

Specifying Ideal Network Port Sources

```
set_ideal_network [get_ports Reset]
compile ...
```

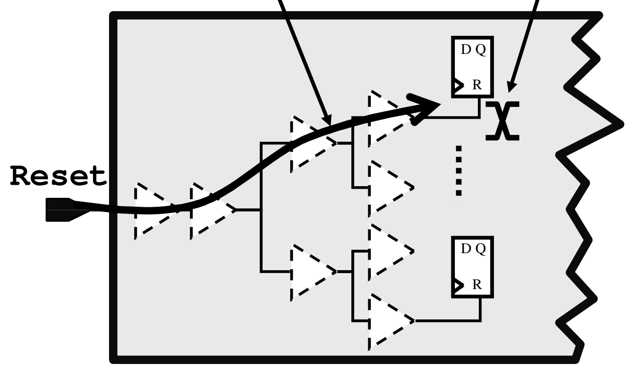


The `set_ideal_network` command applies an *ideal network* attribute to the specified *source* pins or ports, the `Reset` port in the example above. This attribute spreads throughout the transitive fanout of the *source* and stops at sequential cells.

Modeling Ideal Network *Delay* and *Transition*

```
set_ideal_network [get_ports Reset]
set_ideal_latency 1.4 [get_ports Reset]
set_ideal_transition 0.3 [get_ports Reset]
compile ...
```

If needed, you can model the estimated *delay* and *transition time* of the ideal network

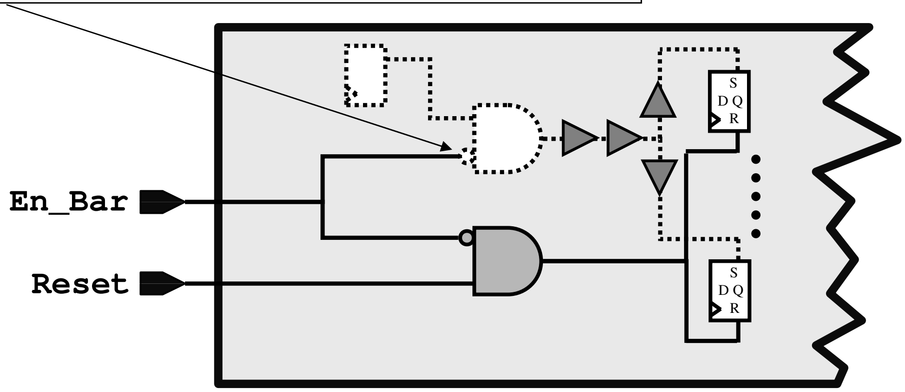


12- 11

Propagation of Ideal Networks

```
set_ideal_network [get_ports {En_Bar Reset}]
compile ...
```

The ideal network stops propagating here



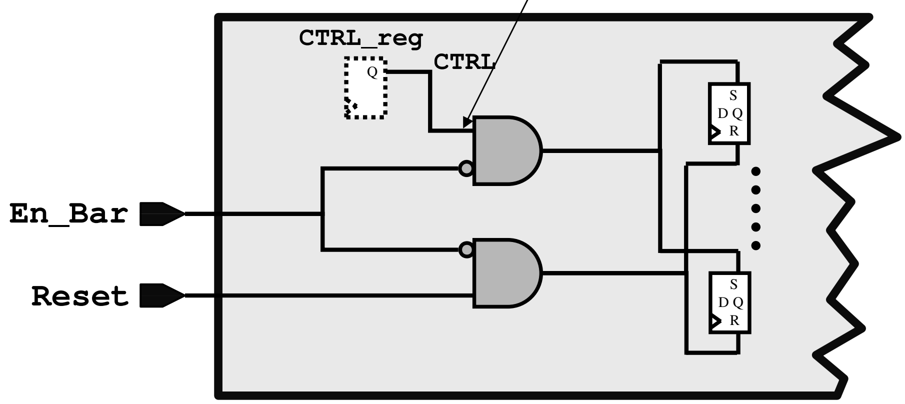
The *ideal network* attribute propagates through a combinational gate only if all input pins of that gate are defined as an *ideal network*

12- 12

Specifying Ideal Network *Pin Sources*

```
set_ideal_network [get_ports {En_Bar Reset}]
set_ideal_network [get_pins CTRL_reg/Q]
OR
set_ideal_network –no_propagate [get_nets CTRL]
compile ...
```

Input pin inherits *ideal network* attribute from ideal network *source pin* `CTRL_reg/Q` --- attribute can now propagate through the logic gate



12- 13

Since both input pins of the top *AND-gate* have an *ideal network* attribute, and at least one of the input attributes is allowed to propagate (does not have a `–no_propagate` option), the ideal network attribute propagates through the AND-gate until it reaches the registers.

In the above example, since the `set_ideal_network` command is applied on the *unmapped* design, prior to compiling it, you will need to know the *GTECH* pin or net names. Here is a summary of the naming conventions: The output pins of *GTECH* registers are `Q` and `QN`, respectively. The data input pin of a *GTECH* flip-flop is `next_state`, while the clock pin is `clocked_on`. The *cell* name of the register is always `REGISTERED_SIGNAL_NAME_reg`. The *net* connected to the flip-flop output is always the `REGISTERED_SIGNAL_NAME`.

A *net* can be specified as the argument but must be accompanied by the `–no_propagate` option. The source of the *ideal network* attribute is the driver pin or port of that net, not the net itself.

Note: In the following example the *SET* network is NOT an ideal network while the *RESET* network is ideal, because both inputs of the top *AND-gate* have the `–no_propagate` option:

```
set_ideal_network [get_ports Reset]
set_ideal_network –no_propagate [get_nets {CTRL En_Bar}]
```

To remove the ideal network source use `remove_ideal_network`.

Unit Agenda

High-fanout net buffering

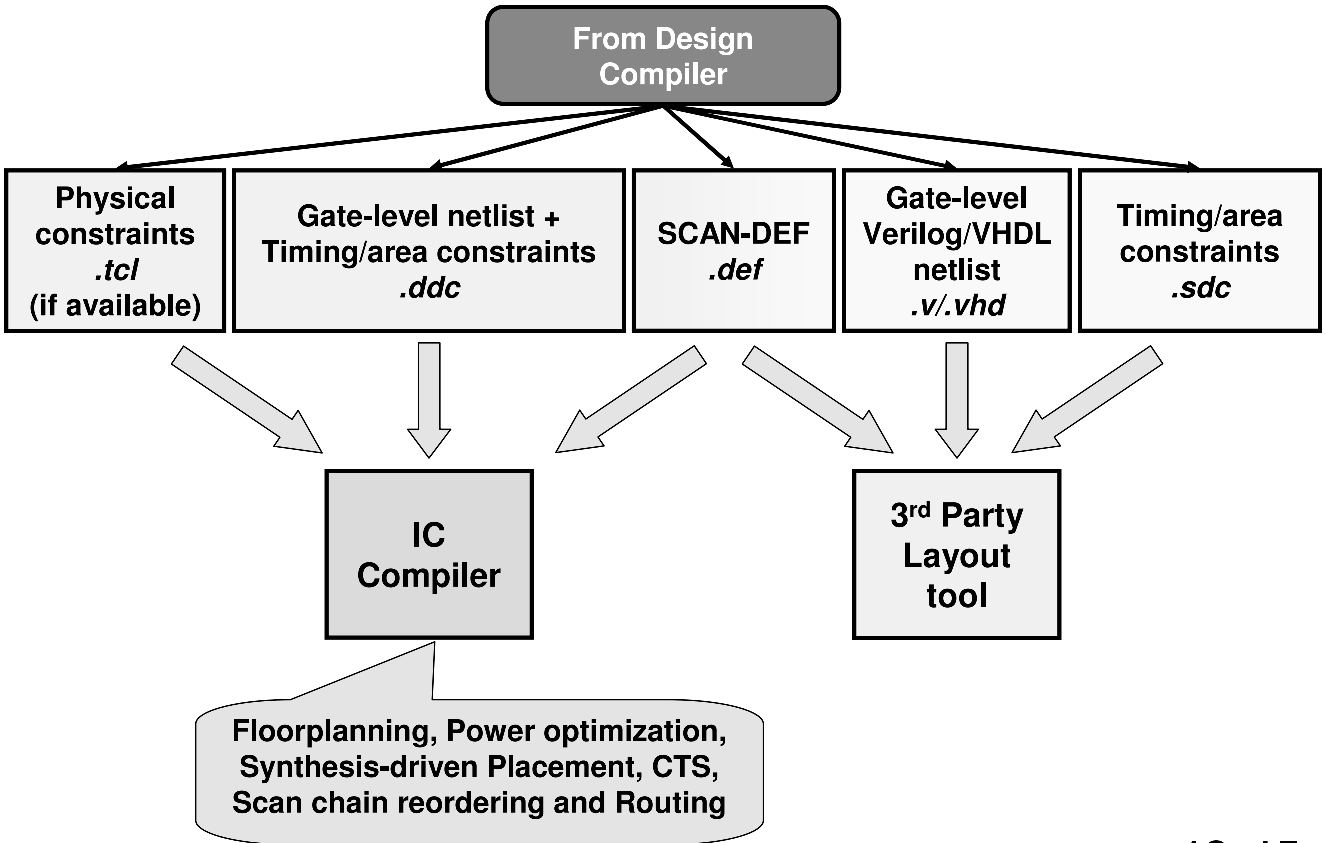
Data needed for physical design

Netlist restrictions

- `assign` statement
- Special characters

12- 14

Data Needed for Physical Design or Layout



12- 15

The *ddc* file contains the gate-level netlist and timing/area constraints. If DFT Compiler was used to insert and stitch up scan chains an additional *SCAN-DEF* file is needed. Lastly, if available from DC topographical mode, the physical constraints file can also be read in.

Instead of reading in the *ddc* file, ICC can also read in a Verilog/VHDL netlist plus the full constraints + directives file (from `write_script`).

Non-Synopsys 3rd party back-end or layout tools do not read in the *ddc* format. They require a standard Verilog/VHDL netlist as input. Since the Verilog/VHDL netlist does not contain any constraints, the constraints are provided as a separate *sdc* format file. The *sdc* file, explained in the next slide, contains a sub-set of the full-blown constraints and directives applied to the design. If these 3rd party layout tools have the ability to re-order scan chains, the *SCANDEF* file should also be provided.

What is *sdc*?

- **3rd party layout tools do not use or understand DC-specific *compile directives* such as:**

```
group_path
set_ungroup
set_cost_priority
set_optimize_registers
set_ultra_optimization
```

- **For these layout tools, write out a 'constraints-only' version of DC's full constraints file[1] with:**

```
write_sdc <my_design.sdc>
```

- **The *sdc* file contains the normal constraints, with expanded[2] arguments**

```
set_max_area; create_clock; set_in/output_delay; set_false_path ...
```

12- 16

[1]The full set of constraints and directives can be written out with
`write_script -out <file_name>`

[2]Besides stripping out compile directives, the `write_sdc` command also 'expands' the command arguments, for example:
The argument `[get_ports sd_* -filter "port_direction == in"]` will be expanded to `[get_ports sd_in[0]]` through `[get_ports sd_in[7]]`.

The argument `[load_of [get_lib_pins max_lib/pc3b03/I]]` will be expanded to its load value, e.g. `0.0325`.

What is *SCAN-DEF*?

- Contains scan chain information
- Used by physical design or layout tools to optimize the grouping and ordering of existing scan chains
- Standard *def* format used by IC Compiler and other 3rd party layout tools

```
write_scan_def -out <my_design.def>
```

12- 17

Recall: Physical Constraints

- **Optional input for `compile_ultra` in *topographical mode***
- **Describe a non-default floorplan for better post-synthesis correlation**
- **Physical constraints can be modified by auto-ungrouping and `change_names` and are not saved in ddc**
- **Write out the physical constraints for IC Compiler prior to 2007.03[1]**

```
write_physical_constraints -output PhysConstr.tcl
```

12- 18

[1] Starting with DC v2007.03 the physical constraints are saved with the *ddc* file, so it is no longer necessary to explicitly write out the constraints for IC Compiler.

Unit Agenda

High-fanout net buffering

Data needed for physical design

Netlist restrictions

- **`assign` statement**
- Special characters

12- 19

Problem: Verilog `assign` Statements

The Problem: Layout tools may not be able to handle `assign` statements in the Verilog netlist

The Solution: Prevent `assign` statements in Verilog netlists caused by:

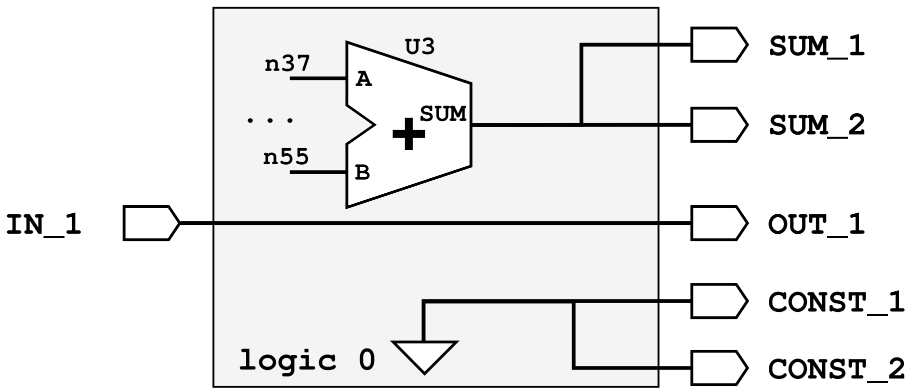
- Multiple port nets
- Verilog *tri* declarations

12- 20

Multiple Port Nets Cause `assign` Statements

Multiple ports or hierarchy pins connected to the same signal, constant nets, as well feed-throughs, use `assign`:

```
input IN_1, ...
output SUM_1, SUM_2, OUT_1, CONST_1, CONST_2 ...
...
DW01_add U3 (.SUM(SUM_1), .A(n37), .B(n55);
assign SUM_2 = SUM_1;
assign OUT_1 = IN_1;
assign CONST_1 = 1'b0;
assign CONST_2 = 1'b0;
```



12- 21

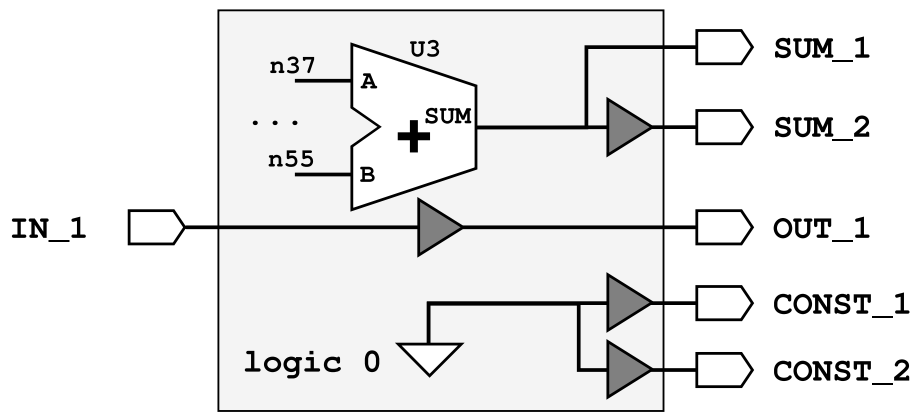
Preventing Multiple Port Nets

To ensure that your final netlist does not contain `assign` statements, separate the multiple port nets during compile:

```
set_fix_multiple_port_nets -all -buffer_constants
compile...
```

Feedthroughs, signals and constants

Buffer constants, instead of duplicating



12- 22

By default `set_fix_multiple_port_nets` is applied to the entire design. You can optionally include a design list.

The code for the above design looks like this – notice – no `assign` staements:

```
DW01_add U3 ( .SUM(SUM_1), .A(n37), .B(n55) );
inv U1 ( .I(SUM_1), .Z(SUM_2) );
inv U2 ( .I(IN_1), .Z(OUT_1) );
inv U4 ( .I(1'b0), .Z(CONST_1) );
inv U5 ( .I(1'b0), .Z(CONST_2) );
```

Verilog `tri` Declarations Cause `assign` Statements

- **DC uses `assign` statements for `tri` signals, but not for `wire` signals – either can be used to model connections with no logic function**
- **Solution: Automatically convert `tri` declarations to `wire` declarations before writing out the netlist**

```
set verilogout_no_tri true
write –f verilog –out ...
```

```
input IN_1, ...
output SUM_1,  OUT_1 ...
reg SUM_1, OUT_1, ...
tri SIG_1, SIG_2;     →  wire SIG_1, SIG_2;
```

12- 23

Unit Agenda

High-fanout net buffering

Data needed for physical design

Netlist restrictions

- `assign` statement
- **Special characters**

12- 24

Special Characters in Netlists

- ***Special characters* in port, cell and net names: Anything other than a number, letter, or underscore**
- **When DC writes out a netlist it inserts backslashes, if needed, to *escape* special characters in port, net and cell names - for example:**
  - The net `bus[7:0]` is expanded into ‘scalar’ names and becomes `\bus[7], \bus[6]...`
    - The brackets in this case are just part of a scalar net name, not a special character denoting the ‘slice’ of a bus
  - VHDL multi-dimensional arrays use square brackets as word subscript delimiters: `\reg[0][19], \reg[0][18]..`
- **<u>The Problem</u>: Layout tools may not recognize DC’s “\” *escape* convention and may therefore misinterpret the special characters**

12- 25

Special Characters Solution: `change_names`

<u>The Solution</u>: Ensure that the netlist is free of special characters by automatically replacing *special characters* with *non-special* ones before writing out the netlist

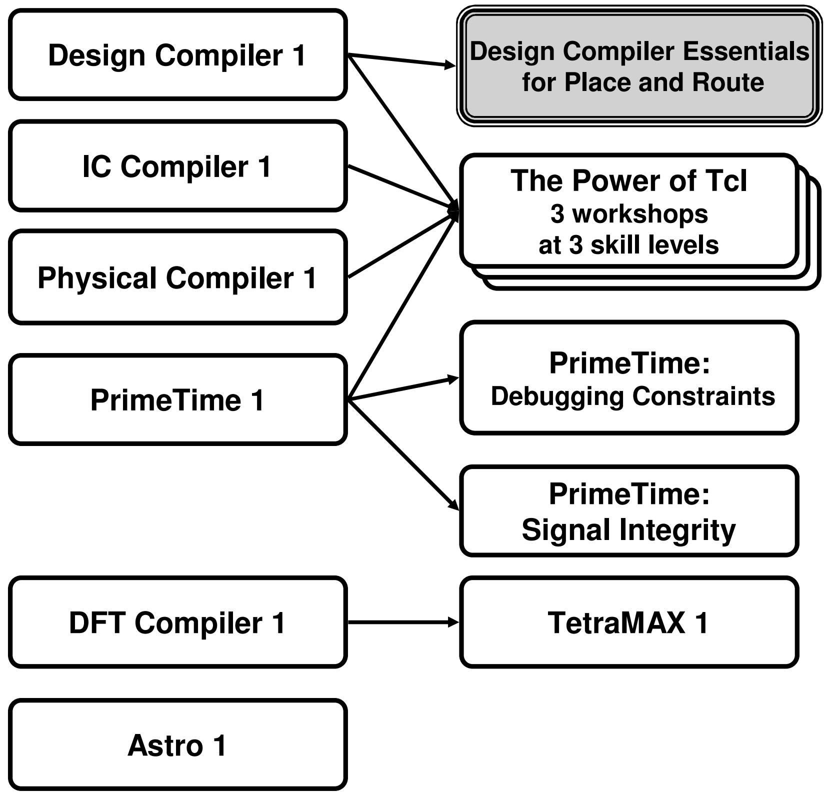
```
change_names -rules verilog -hier
write -f verilog -out ...
```

```
\bus[7]        →   bus_7_
\reg[0][19]    →   reg_0__19_
```

12- 26

Note: The `verilog` rules files can handle most common Verilog and VHDL netlist changes. It is possible to create customized “rules” for special needs.

For More Information …



12- 27

Summary: *Ideal Network* Commands

```
create_clock -period 2.5 [get_ports clk]
set_clock_uncertainty -setup 0.3 [get_clocks clk]
set_clock_transition 0.2 [get_clocks clk]
...
# Disable timing/DRC optimization of HFN Port sources
set_ideal_network [get_ports reset* select*]

# Disable timing/DRC optimization of HFN Pin sources if
# GTECH pin names and/or net names are known
set_ideal_network [get_pins FF_SET_reg/Q]
set_ideal_network -no_propagate [get_nets CTRL]

# Optional replacement values for default zero delay and
   transition values of ideal networks
set_ideal_latency 1.4 [get_ports reset* select*]
set_ideal_transition 0.5 [get_pins FF_SET_reg/Q]
```

12- 28

For scan-enable can't use `set_ideal_network` because this will prevent `insert_dft` from hooking up the scan-enable signals.

Summary: *Output Data* Commands

```
...
# Insert buffers for all multiple-port nets – eliminate assign
set_fix_multiple_port_nets -all -buffer_constants
compile ...  or compile_ultra ...
# Convert tri to wire - eliminate assign
set verilogout_no_tri true
# Eliminate special characters in the netlist and constraints file
change_names -rules verilog -hierarchy
write -f ddc -hierarchy -output my_ddc.ddc
write -f verilog -hierarchy -output my_verilog.v
# Write out the constraints-only sdc file
write_sdc my_design.sdc
# Write out the scan chain information
write_scan_def -out my_design.def
# Write out the physical constraints
write_physical_constraints -output PhysConstr.tcl
```

12- 29

Note: `set_fix_multiple_port_nets` must be applied before compiling the design, because the fix happens during *compile* – it is recommended to do so before the first *compile*.

The `set verilogout_no_tri true` and `change_names` commands should be applied after all optimizations, just prior to writing out the netlist and/or constraints.

Unit Objectives Review

You should now be able to:

- **Generate required post-synthesis output data files that take into account the abilities and restrictions of downstream physical design tools**
- **Select the appropriate workshop to learn more about physical design tool considerations**

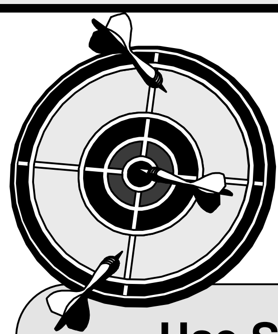
12- 30

Agenda

DAY 3

Synopsys 10-I-011-SSG-013 13- 1

Recall: Workshop Goal



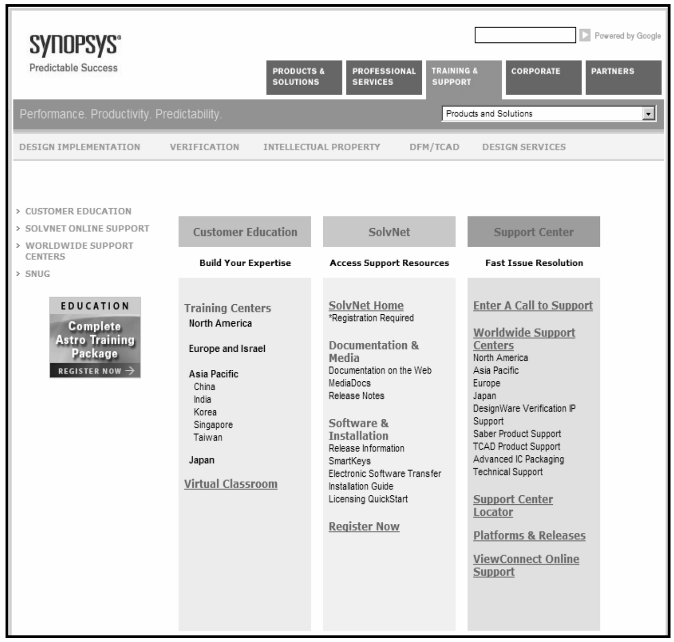
Use Synopsys' Design Compiler to:

- **Constrain a complex design for area and timing**
- **Apply synthesis techniques to achieve area and timing closure**
- **Analyze the results**
- **Generate design data that works with physical design or layout tools**

13- 2

Synopsys Support Resources

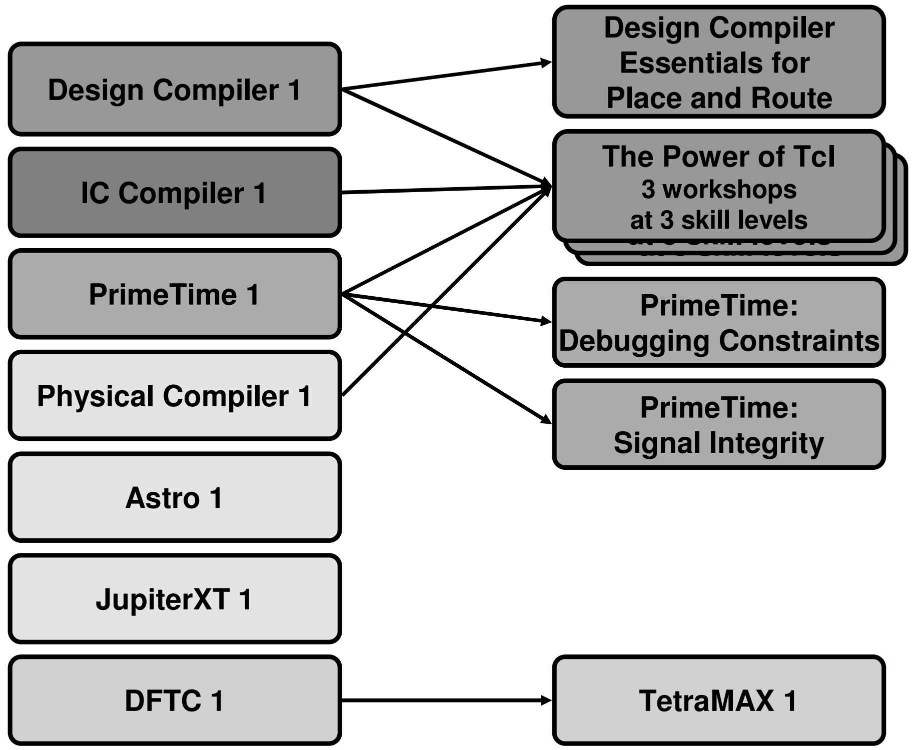
1. **Build Your Expertise: Customer Education Services**
   www.synopsys.com/services/education
   - Workshop schedule and registration
   - Download materials
2. **Empower Yourself: solvnet.synopsys.com**
   - Online technical information and access to support resources
   - Documentation & Media
3. **Access Synopsys Experts: Support Center**



13- 3

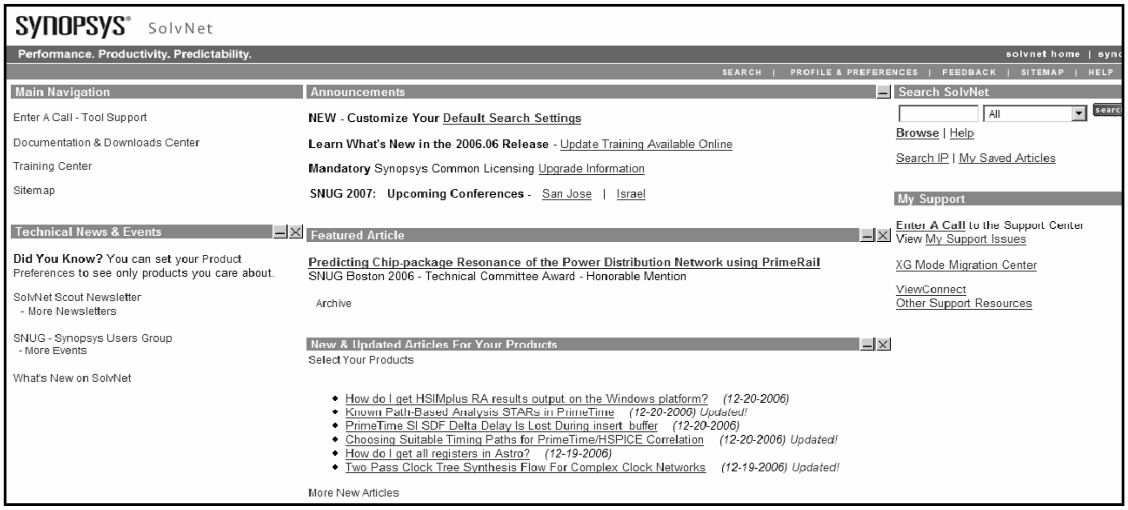
Workshop Curriculum

Register at 1-800-793-3448 or www.synopsys.com/services/education



13- 4

SolvNet Online Support Offers

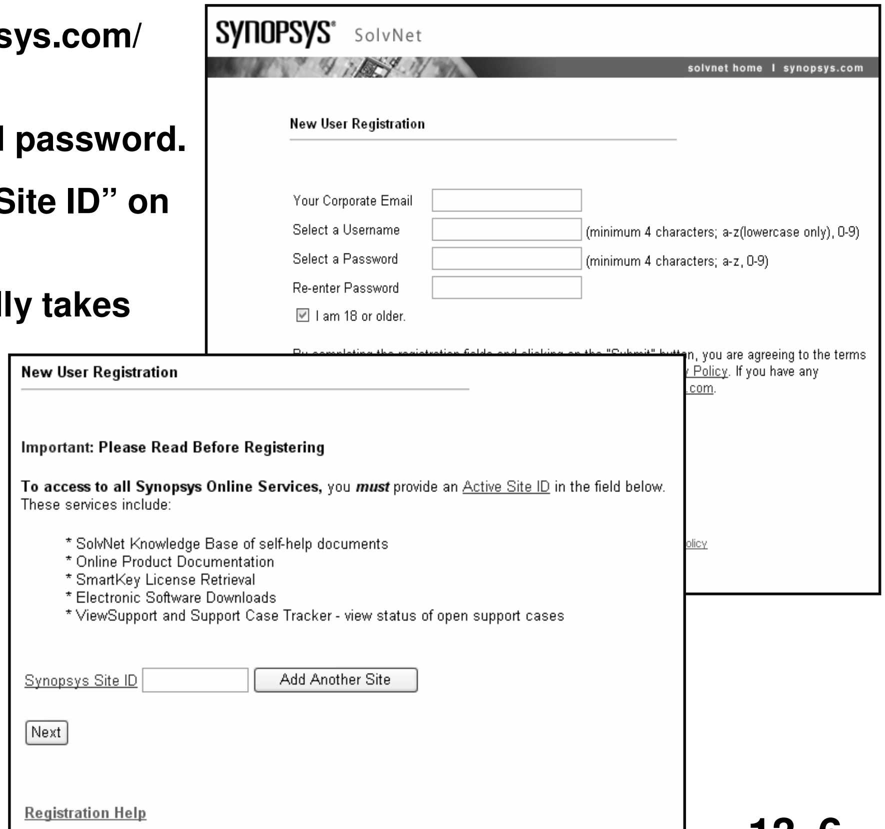


- **Immediate access to the latest technical information**
- **Thousands of expert-authored articles, Q&As, scripts, tool tips**
- **Online Support Center access: *Enter A Call –Tool Support***
- **Release information**
- **Electronic software downloads**
- **Synopsys announcements (latest tool, event and product information)**
- **Online documentation**
- **License keys**

13- 5

SolvNet Registration is Easy

1. Go to solvnet.synopsys.com/ ProcessRegistration
2. Pick a username and password.
3. You will need your “Site ID” on the following page.
4. Authorization typically takes just a few minutes.



13- 6

Support Center: AE-based Support

- **Industry seasoned Application Engineers:**
  - 50% of the support staff has > 5 years applied experience
  - Many tool specialist AEs with > 12 years industry experience
  - Access to internal support resources
- **Great wealth of applied knowledge:**
  - Service >2000 issues per month
- **Remote access and debug via ViewConnect**

Fastest access

- **Contact us:**
  - **Web: *Enter A Call* from solvnet.synopsys.com**
  - **See http://www.synopsys.com/support for local support resources**

13- 7

Other Technical Sources

- **Application Consultants (ACs):**
  - Tool and methodology pre-sales support
  - Contact your Sales Account Manager for more information
- **Synopsys Professional Services (SPS) Consultants:**
  - Available for in-depth, on-site, dedicated, custom consulting
  - Contact your Sales Account Manager for more details
- **SNUG (Synopsys Users Group):**
  - www.snug-universal.org

13- 8

Summary: Getting Support

- Customer Education Services
- SolvNet
- Support Center
- SNUG

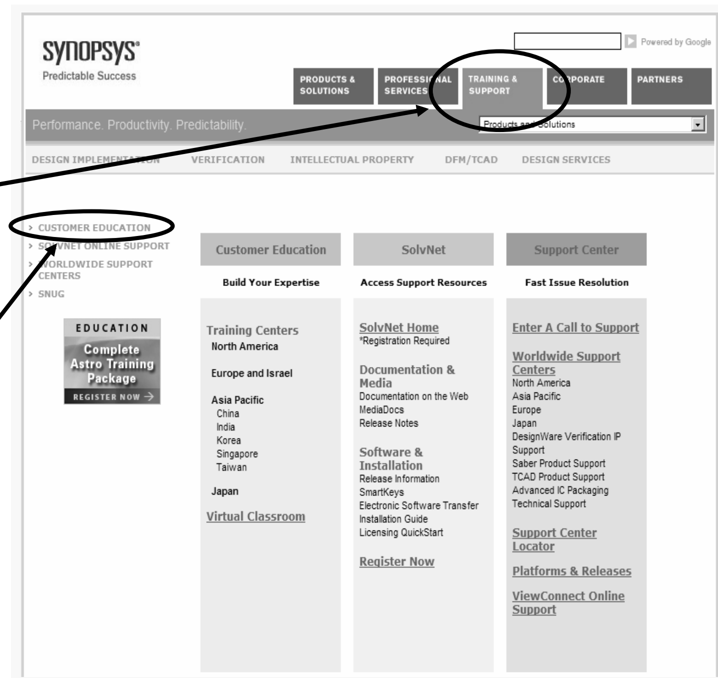
13- 9

How to Download Lab Files (1/4)

From the Synopsys home page:
http://www.synopsys.com

Select the
TRAINING & SUPPORT tab

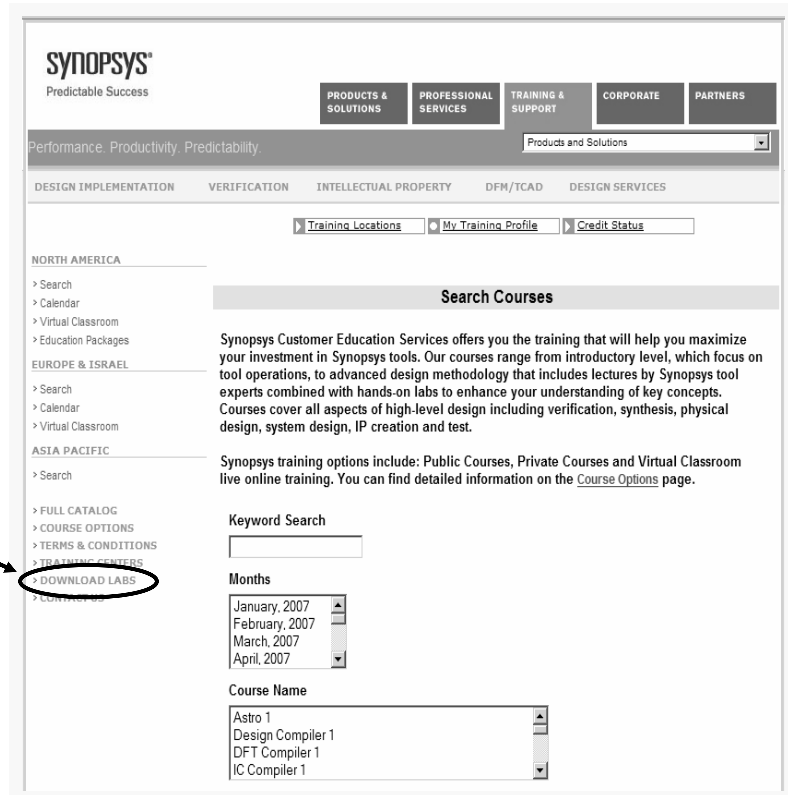
Select
CUSTOMER EDUCATION



13- 10

How to Download Lab Files (2/4)

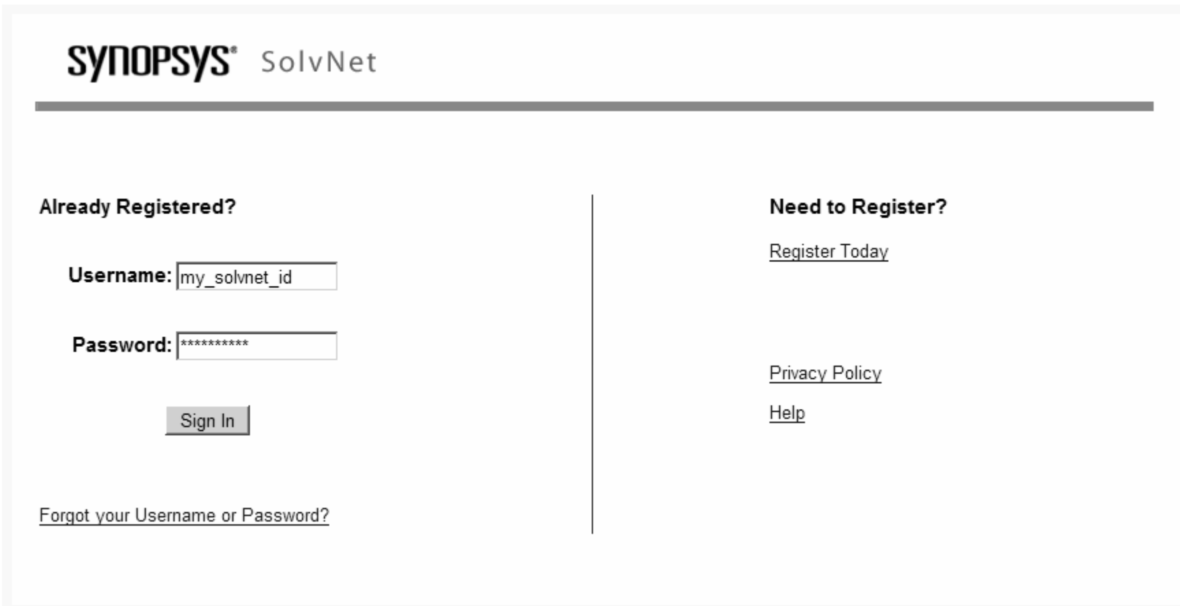
Select DOWNLOAD LABS



13- 11

How to Download Lab Files (3/4)

Login to SolvNet



13- 12

How to Download Lab Files (4/4)

Locate Course Title

Workshop Description	Software Version	FTP Product Directory
Astro 1	2005.09	Labs_ASTRO1_2005.09 Download
stro Rail - Power Integrity Analysis	2004.12	Labs_ASTRO_RAIL_2004.12 Download
stro: Advanced Clock Tree Synthesis	2004.12	Labs_ASTRO_ACTS_2004.12 Download
C FPGA Synthesis	2004.06	Labs_DCFPGA_2004.06 Download
Design Compiler 1	**2007.03**	**Labs_DC1_2007.03** Download
Design Compiler Essentials for Place and Route	2005.09-SP3	Labs_DCEPnR_2005.09-SP3 Download
DFT Compiler 1	2006.06	Labs_DFTC1_2006.06 Download
Formality	2004.06	Labs_FORMALITY_2004.06 Download
Hercules Beginning Runset	2004.12	Labs_HERCULES_RUNSET_2004.12 Download

Click to download the *.tar.gz* file.

(You may be asked for your *SolvNet* password a second time)

A *README* file will walk you through the *decompression* and *untar* steps

13- 13

That's all Folks!

13- 14