95-1 Under-Graduate Project
Synopsys Synthesis Overview

Speaker: 陳 彥 良
Adviser: Prof. An-Yeu Wu
Date: 2006/11/22
Outline

- Introduction
- Synopsys Graphical Environment
- Setting Design Environment
- Setting Design Constraints
- Synthesis Report and Analysis
What is Synthesis (1/2)

- Synthesis = translation + optimization

```verilog
always @(reset or set)
begin : direct_set_reset
  if (reset)
    y=1'b0;
  else if (set)
    y=1'b1;
end

always @(gate or reset)
  if (reset)
    t=1'b0;
  else if (gate)
    t=d;
```

![Diagram showing the process of translation and optimization]
What is Synthesis (2/2)

- Synthesis is Constraint Driven
- Technology Independent
Logic Synthesis Overview

- RTL Design
- HDL Compiler
- Design Compiler
- Design Library
- Technology Library
- DW Developer
- Lib Compiler

Architecture Optimization

Logic Optimization

optimized
Gate-level Netlist

no timing info.

timing info.
# Synopsys Related Files

<table>
<thead>
<tr>
<th>Files</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cshrc</td>
<td>Set path and environment variables and license check</td>
</tr>
<tr>
<td>.Xdefault</td>
<td>Set X terminal display variables</td>
</tr>
<tr>
<td>.synopsys_dc.setup</td>
<td>Three distinct files are read and executed when DC is invoked</td>
</tr>
<tr>
<td></td>
<td>1. system-wide (do not modify):</td>
</tr>
<tr>
<td></td>
<td>(e.g. $SYNOPSIS/admin/setup/)</td>
</tr>
<tr>
<td></td>
<td>2. User’s home directory (e.g. ~think/)</td>
</tr>
<tr>
<td></td>
<td>3. User’s current working directory (e.g. ~think/dv/)</td>
</tr>
</tbody>
</table>

- **Note**
  - These 3 files are always read *in the same order*.
  - Any repeated command can *override* the previous one.
## Tools We will Use

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design Vision</strong></td>
<td>User Graphical Interface of synopsys synthesis tool</td>
</tr>
<tr>
<td><strong>HDL Compiler</strong></td>
<td>Translate Verilog descriptions into Design Compiler</td>
</tr>
<tr>
<td><strong>Design Compiler</strong></td>
<td>Constraint driven logic optimizer</td>
</tr>
<tr>
<td><strong>Design Time</strong></td>
<td>Static Timing Analysis (STA) engine</td>
</tr>
<tr>
<td><strong>DfT Compiler</strong></td>
<td>Design for Testing</td>
</tr>
<tr>
<td><strong>Design Ware</strong></td>
<td>Enable synthesis using DesignWare library</td>
</tr>
</tbody>
</table>
Design Vision
HDL Compiler

HDL Compiler translates Verilog HDL descriptions into Design Compiler as Synopsys design block

always @ (reset or set) begin : direct_set_reset
  if (reset)
    y=1'b0;
  else if (set)
    y=1'b1;
end
always @(gate or reset) begin : gate_set
  if (reset)
    t=1'b0;
  else if (gate)
    t=d;
end
HDL Compiler

- In schematic view, we can see the Verilog file is translated with a GTECH library (the synopsys default)
Design Compiler

- Design Compiler maps Synopsys design block to gate level design with a user specified library

![Design Compiler Interface](image)
Design Compiler Interaction

Three ways to interface

- **dc_shell** (Legacy Interface)
  - `dc_shell –t`
- **dc_shell** (TCL Interface)
- **Design Vision** (GUI)
  - `dv` with `-dcsh_mode`

Command line

**dc_shell**
## Synopsys Related Files

<table>
<thead>
<tr>
<th>Files</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.cshrc</code></td>
<td>Set path and environment variables and license check</td>
</tr>
<tr>
<td><code>.Xdefault</code></td>
<td>Set X terminal display variables</td>
</tr>
<tr>
<td><code>.synopsys_dc.setup</code></td>
<td>Three distinct files are read and executed when DC is invoked</td>
</tr>
<tr>
<td></td>
<td>1. system-wide (do not modify):</td>
</tr>
<tr>
<td></td>
<td>(e.g. $SYNOPSYS/admin/setup/)</td>
</tr>
<tr>
<td></td>
<td>2. User’s home directory (e.g. ~think/)</td>
</tr>
<tr>
<td></td>
<td>3. User’s current working directory (e.g. ~think/dv/)</td>
</tr>
</tbody>
</table>

### Note

- These 3 files are always read *in the same order*.
- Any repeated command can *override* the previous one.
Synopsys On-Line Documentation (SOLD)

- Invoke Synopsys On-Line Document using the command
  
  `unix%> acroread /usr/synopsys/sold/cur/top.pdf`

- Note: whenever you find a question, check SOLD first
What .synopsys_dc.setup defined

- **link_library**: the library used for interpreting input description
  - Any cells instantiated in your HDL code
  - Wire Load or Operating Condition models used during synthesis
- **target_library**: the ASIC technology that the design is mapped to
- **symbol_library**: used during schematic generation
- **search_path**: the path to search for unsolved reference library or design
- **synthetic_library**: designware library to be used
- Other variables
In CIC cell_based flow, we support compass 0.18um cell library, the .synopsys_dc.setup file is as follows

```plaintext
search_path =.{ /your_path/CBDK018_TSMC_Artisan//CIC/SynopsysDC}+search_path;
set link_library "slow.db fast.db dw.Foundation.sldb"
set target_library "slow.db fast.db"
set symbol_library "generic.sdb"
set synthetic_library "dw.Foundation.sldb"

set bus_inference_style {%s[%d]}
set bus_naming_style {%s[%d]}

set hdlin_translate_off_skip_text "TRUE"
set edifout_netlist_only "TRUE"
set verilogout_no_tri true
set plot_command {lpr -Plp}
set view_script_submenu_items [list \ {Avoid assign statement} {set_fix_multiple_port_nets -all -buffer_constant} \ {Change Naming Rule} {change_names -rule verilog -hierarchy} \ {Write SDF} {write_sdf -version 2.1 -context verilog chip.sdf}]
```
Synthesis Design Flow

- Develop the HDL design description and simulate the design description to verify that it is correct.
- Set up the .synopsys_dc.setup file.
  - Set the appropriate technology, synthetic, and symbol libraries, target libraries, and link libraries.
  - Set the necessary compilation options, including options to read in the input files and specify the output formats.
- Read the HDL design description.
- Define the design.
  - Set design attributes
  - Define environmental conditions
  - Set design rules
  - Set realistic constraints (timing and area goals)
  - Determine a compile methodology
Synopsys Graphical Environment
Invoke Design Vision

- Unix%> dv &

![Screen capture of Design Vision interface]

---

dc_shell command
Optimization Using the Design Vision

- File/Analyze & File/Elaborate - Verilog & VHDL, or File/Read - all other formats
- Attributes - set up Design Environment & Goals
- Analysis/Report - check if set up is OK
- Analysis/Check Design
- Tools/Design Optimization
- Analysis/Report
- File/Save
Analyze & Elaborate

- Use **analyze** and **elaborate** to bring Verilog or VHDL files into design compiler memory.

- **Analyze** does **syntax checking** and produces an **intermediate .syn .mra files** to be stored in a design library.

- **Elaborate** looks in the design library for the .syn file and builds the design up into design compiler memory (as **design block**).
Analyze

- Check VHDL & Verilog for syntax and synthesizability
- Create intermediate .syn and .mra files and places them in library specified – design library

- Equivalent to dc_shell
  ```
  analyze -format verilog -library WORK counter.v
  ```
Elaborate

- Elaborate after analyze to bring design into Design Compiler memory using generic components (GTECH)
- Look in the design library for intermediate .syn file for design specified

```
elaborate counter -architecture verilog -library WORK -update
```

File/Elaborate
Read File

- Read netlists or other design descriptions into Design Compiler
- File/Read
- Support many different formats:
  - synopsys internal formats
  - DB(binary): .db
  - equation: .eqn
  - state table: .st
  - Verilog: .v
  - VHDL: .vhd
  - PLA(Berkeley Espresso): .pla
  - EDIF

![Read Designs Window](image)
Four Types of Icon

- **EQUATION**
  - Equation, or non-netlist VHDL or Verilog format

- **PLA**
  - Programmable logic array format

- **FSM**
  - Finite-state-machine design represented as a state table

- **NETLIST**
  - Design was read in as a netlist (including structural VHDL and Verilog), or it has been optimized
Describe the Design Environment

- You can use Design Vision to constrain your design
Compile the Design

- The compile command optimizes and maps the current_design
Report the Design

- From report and analysis, you can find the set attributes and the results after optimization
Save the Design
Check Design

- **Design/Check Design**
- Execute `check_design` before you optimize your design
- Two types of messages are issued
  - **error**
    - Error: In design ‘bcd7segs’, cell ‘decoder’ has more pins than it’s reference ‘d1’ has ports
  - **warnings**
    - Warning: In design ‘converter’, port ‘A’ is not connected to any nets
Different View - Design View

Hierarchy
Schematic
Symbol
Current Design Indicator
View Indicator
Setting Design Environment
Setting Design Environment

- Setting Operating Environment
- Setting Input Driving Strength
- Setting Output Loading
- Setting Input/Output Delay
- Setting Wire Load Model
Setting Operating Environment

- Attributes/Operating Environment
Setting Operating Condition

- Attributes/Operating Environment/Operating Condition
  - `dc_shell> set_operating_conditions “slow”`
Setting Input Drive Impedance

- Attribute/Operating Environment/Drive Strength
Setting Input Drive Impedance

- Also can be set using “drive_of” command
- Example: (P2A cell output)
Setting Output Loading

- Attribute/Operating Environment/Load
Setting Output Loading

- Also can be set using load_of:
  - Example: (bufferd1 cell input)
Setting Input Delay

- Select input ports
- Attributes/Operating Environment/Input Delay

Specified clock list

Specify relative clock

Example

```
dc_shell> set_input_delay -clock clk -max 6.4 in1
dc_shell> set_input_delay -clock clk -min 4.4 in1
```
Setting Output Delay

- Select output ports
- Attributes/Operating Environment/Output Delay

![Diagram showing output ports and delay settings]

- Example

```
dc_shell> set_output_delay -clock clk -max 5.3 out1
```
Setting Wire Load Model

- Wire load model estimates wire capacitance based on chip area & cell fanout
- Setting this information during compile in order to model the design more accurately
- Attributes/Operating Environment/Wire Load
Setting Design Constraints
Constraints

- Constraints are goals that the Design Compiler uses for optimizing a design into target technology library.
- **Design Rule Constraints**: technology-specific restriction; ex. maximum transition, maximum fanout, maximum capacitance.
- **Optimization Constraints**: design goals and requirements; ex. maximum delay, minimum delay, maximum area, maximum power.
- During compile, Design Compiler attempts to meet all constraints.
Setting Design Constraints

- **Optimization Constraints**
  - Basic clock constraints concept
  - Constraints for Area

- **Design Rule Constraints**

- **Final check constraints before compile**
Define Clock Specification

- What should be defined?
  - Period
  - Waveform
  - Uncertainty
    - Skew
  - Latency
    - Source latency (option)
    - Network latency
  - Transition
    - Input transition
    - Clock transition
Specify Clock Constraints

- Select clock port
- Attributes/Clocks/Specify

- *creat_clock*: define your clock’s waveform & respect the set-up time requirements of all clocked flip-flops
  - *creat_clock* “clk”-period 50 - waveform \{0 25\}

- *set_fix_hold*: respect the hold time requirement of all clocked flip-flops
  - *set_fix_hold* clk

- *set_dont_touch_network*: do not re-buffer the clock network
  - *set_dont_touch_network* clk
Setting Area Constraint

- Attributes/Optimization Constraints/Design Constraints
- Area Unit: gates
Setting Design Constraints

- Optimization Constraints
  - Basic clock constraints concept
  - Constraints & STA for Special Circuit
  - Constraints for Area

- Design Rule Constraints

- Final check constraints before compile
Design Rule Constraints

- Design rules can’t be violated at any cost, even if it will violate the timing and area goal.
- Three kinds of design rule constraint are set:
  - `set_max_transition`
  - `set_max_fanout`
  - `set_max_capacitance`
Setting Design Constraints

- Optimization Constraints
  - Basic clock constraints concept
  - Constraints & STA for Special Circuit
  - Constraints for Area

- Design Rule Constraints

- Final check constraints before compile
Check Design

- After you set up the design attributes & design constraints, we recommend the next step is to check design

- Analysis/Check Design

- The warning message is called "multiple design instance", it results from that you use the same HDL description to represent more than one design instance

- How to handle?
  - dont_touch
  - ungroup
  - uniquify
Method 1: dont_touch

- Procedures
  - Constrain the block
  - Compile the block
  - Select the multiple design instances block
  - Attributes/Optimization Directives/Design & set the Don’t Touch button
  - Compile the whole design using hierarchy compile
Method 2: Ungroup

- Procedures
  - Select the multiple design instances block
  - **Attributes/Optimization Directives/Design** & set the Ungroup button
  - Compile whole design using hierarchy compile
- Remove a single level of hierarchy
- Does not preserve the hierarchy
- Take more memory
- Take more compile time
Method 3: Uniquify

- Create a unique design file for each instance
- May select one cell or entire design hierarchy to be uniquify
- Allow design to be customized to its interface
- If the environment varies significantly, use uniquify rather than compile+dont_touch
- Uniquify uses more memory and cause longer compile time than compile+dont_touch

Select the most top design of the hierarchy

Edit/Uniquify/Hierarchy
Multiple Design Instance (summary)

- Use “
dont_touch, ungroup, uniquify” to fix it
- The easiest way is 
uniquify, but needs much memory & compile time
- If you want to preserve the hierarchy & source sharing, use 
don’t_touch
- If you want your design to have the 
BEST result, recommend to use 
ungroup, but it needs the most memory and compile time
Setting Design Constraints

- Optimization Constraints
  - Basic clock constraints concept
  - Constraints & STA for Special Circuit
  - Constraints for Area

- Design Rule Constraints

- Final check constraints before compile
Check Constraints & Attributes

- Use the following reports to check constraints & attributes before compiling
- Analysis/Report
Save Constraints & Attributes

- Save attributes & constraints setting as the design setup file in dc_shell command format, use File/Save Info/Design Setup

Output File Name: top_setup.dc
Execute Script File

- Execute dc_shell command script file, use Setup/Execute Script
Synthesis Report and Analysis
Report

- **Analysis/Report**
  - From report and analysis, you can find the set attributes and the results after optimization

- **Attribute reports**
  - All attributes, clock, port, design, net

- **Analysis reports**
  - Area, hierarchy, constraints, timing, point timing
Timing Report

ClockSkew

Net_delay + cell_delay are combined

Net_delay

C.C.YANG/2001
Analyze Circuit with Schematic

- To determine the time the signal arrived at pin which is selected in the schematic
- Analysis/Show Timing
Analyze Circuit with Schematic

- To determine the net load which selected in the schematic
- Analysis/Show Net Load
Save Design

- Save your design in verilog format, run Verilog gate-level simulation, and we will use Verilog In interface to translate it into OPUS database for place & route
- If you can’t Verilog In, please check assign problem
- If there is any assignment problem, choose the block & use the dc_shell command as follow to fix it
  - set_fix_multiple_port_nets -all -buffer_constants
  - compile -map_effort medium

```verilog
assign ABSVAL[18] = `A[18];
assign ABSVAL[17] = `A[17];
assign ABSVAL[16] = `A[16];

bufda X37X (.i(A[19]), .z(ABSVAL[19]));
bufdd X38X (.i(A[18]), .z(ABSVAL[18]));
bufdd X39X (.i(A[17]), .z(ABSVAL[17]));
bufdd X40X (.i(A[16]), .z(ABSVAL[16]));
bufdd X41X (.i(A[15]), .z(ABSVAL[15]));
```
Gate-Level Simulation (Verilog)

- Write out gate-level netlist
  - File/Save As ➔ Verilog (for File format)
  - `dc_shell> write -format verilog -hierarchy -output chip.vg`
- Get SDF
  - File/Save Info ➔ Design timing ➔ Select chip.sdf
  - `dc_shell> write_sdf –version 2.1 -context verilog chip.sdf`
- Modify your testbench file
  - `$sdf_annotate ("the_SDF_file_name", top_module_instance_name);`
- Simulation using Verilog-XL
  - `>> ncverilog chip.vg testbench.v –v cell_model.v +access+r`
Design Example
Synopsys Design Vision (GUI) / Design Compiler (text mode)

- Unix% dv –dcsh_mode &
- Unix% dc_shell
Read Verilog File

read -format verilog "{"Lab1_alu.v"}"
Schematic view

- Synopsys Design analyzer will translate verilog code into G-tech model. Double click the icon “ALU”, and click the right button then choose Schematic view. We can get the G-tech MAP
Symbol view

- Or you can create a symbol view by clicking on the following symbol view button. The symbol view is as the right window.
Set Clock (1/2)

“Attributes”-“Specify Clock”
Set Clock (2/2)

- Specify the clock as period 10ns. (100 MHz). Don’t forget to select “**don’t touch network**” and “**fix hold**”

```bash
create_clock -name "clk" -period 10 -waveform {"0" "5"} {"clk"}
set_dont_touch_network find( clock, "clk")
set_fix_hold clk
```
Operating Condition

- set_operating_conditions "typical" -library "typical"
Wireload Model

- set_wire_load_model -name "ForQA" -library "typical"
- set_wire_load_mode "segmented"
Operating Environment

Select “inputA” in the Symbol View and click “Attribute”-“operating environment”-“input delay”. Set 2.5ns input delay.

set_input_delay -clock clk 2.5 inputA[*]
set_input_delay -clock clk 3.8 inputB[*]
set_input_delay -clock clk 4.5 instruction[*]
set_input_delay -clock clk 5.2 reset

set_output_delay -clock clk 8 alu_out[*]
Click "Attribute"-"optimization Constraints"- "Design constraints". Set max area is 0. Max fan-out is 8. max transition is 1.

```
set_max_area 0
set_max_fanout 8 find (design, ALU)
set_max_transition 1 find (design, ALU)
```
Compile Design

- Click “Design” - “Compile Design”. Click “OK”, start to optimize ALU

```
compile -map_effort medium
```
Report

```
report_timing -path full -delay max -max_paths 1 -nworst 1
report_power
report_area -nosplit
```
Save Files

- Save gate-level netlist. Select “File”->”Save As”
- Save your design. Select “File”->“Save”
- Save the timing information. Select “File”->“Save Info”->”Design Timing”, choose sdf format.

write -format verilog -hierarchy -output "ALU_s.v" find (design, ALU)
write -format db -hierarchy -output "ALU_s.db" find (design, ALU)
write_sdf  ALU_s.sdf

- Save script file with the constraints you have made. Use “write_script > script_file” command or “File”->”Save Info”->”Design Setup“ button.
- Re-run all steps automatically. Use “include script_file” command or “File”->“Execute Script” button.
gate level simulation

- Before gate level simulation, 
  \$sdf\_annotate("top\_design.sdf", top\_design) must be added after initial in testbench
- `\timescale 1ns/10ps` must be added in the 1st line of testbench
- //RTL simulation
- Unix% `ncverilog testfixture.v your_file.v +access+r`
- //gate level simulation
- Unix% `ncverilog testfixture\_vg.v your_file.vg –v tumc18.v +access+r`
View wave form

- Unix% nWave&
- Open verilog.dump file to see the waveform