VLSI Design Flow

Specifications & System Simulation (MATLAB, Simulink, C++)

RTL Design (Verilog HDL)

Logic Synthesis (synopsys DC)

PnR (cadence SoC Encounter)

DRC & LVS (Mentor Graphics Calibre)

Today
Logic Synthesis

- Conceptual overview & tool setup
  - RTL modification
  - Technology libraries
  - Design environment & constraints
  - Major synthesis commands
  - Gate-level simulation

Synthesis:
WHY, HOW, WHAT
**Why Synthesis?**

- Quick (re)design time
- Separate functionality, technology-dependent parameters, and design constraints
- Fast timing/area/power estimates

**How to do Synthesis?**

RTL Design (your .v files) → Architectural Mapping → Logic Optimization → Optimized Gate-level Netlist

Synthesis Commands (your .tcl files) → Designware Lib → Technology Lib

- RCA, CLA...
- Database of Adders, Multipliers, Shifters...
- Gate sizing, Logic Transformation
- MUX
- AND + OR + INV...
What can Synthesis Provide?

- Timing/Area/Power reports
- Gate-level netlist (.vg) & timing info. in standard delay format (.sdf) for timing-aware simulation

```markdown
assign C = A + B;
```

before synth. Glitch & Delay after synth.

---

Script Example (.tcl Files)

```tcl
remove_design -all
source ./SET_CON/T40GP_LibSetup.tcl
lappend search_path ./HDL ./SYN
analyze -format verilog Adder.s
set PROCESS_T40GP
set DESIGN_NAME ADD
elaborate $DESIGN_NAME
link
set_operating_conditions -min ff1p16vn40c -max ss0p95v125c
set TCLK 1.9
set TCU 0.1
source ./SET_CON/T40GP_VarSetup.tcl
set_false_path -from [get_ports ACLR]
uniquify
compile -only_design_rule
compile Ultra-no_autoungroup
compile -inc-only_hold_time
set_fix_multiple_port_nets -all -buffer_constants [get_designs *]
remove_unconnected_ports -blast_buses [get_cells -hier]
report_timing -path full -delay min -max_paths 10 > $LOGPATH$TOPLEVEL$PROCESS.holdtiming
report_timing -path full -delay max -max_paths 10 > $LOGPATH$TOPLEVEL$PROCESS.setuptiming
report_area -hierarchy > $LOGPATH$TOPLEVEL$PROCESS.area
report_power -hier -hier_level 2 > $LOGPATH$TOPLEVEL$PROCESS.power
```
Tool Setup on Workstation

Set path = (/w/apps3/Synopsys/Design_Compiler/vF-2011.09/bin $path)
Source /w/apps3/Synopsys/Design_Compiler/vF-2011.09/SETUP

1. Login eeapps.seas.ucla.edu
2. Edit .cshrc file, or directly type
3. Type dc_shell to open DC

Now you’re ready to use **Synopsys Design Compiler**

Logic Synthesis

- Conceptual overview & tool setup
- **RTL modification**
  - Technology libraries
  - Design environment & constraints
  - Major synthesis commands
  - Gate-level simulation
The Importance of HDL Coding Styles

Different coding styles that are functionally equivalent may be mapped into HWs having different timing/area.

\[ E = A + B + C + D \quad \text{E} = (A + B) + (C + D) \]

Good coding style: Better starting point for DC to reach optimal gate-level results.

Enable Cool Features from DC

#1: Gated Clock
#2: Synopsys Directives
Go Low-Power using **Clock Gating**

30~50% Power Savings @ Block Level

**STILL** toggles when EN = 0

**NO** toggle when EN = 0

always@(posedge CLK)
if(EN) Q <= D ;
else Q <= Q ;

Two steps for gated-clock syn.: (1)Delete **Else** statements in seq. logic; (2)DC commands (shown later)

---

**Case Directives Supported by DC (1)**

tell DC to interpret incomplete case specially

Keyword:

```
// synopsys_parallel_case
// synopsys_full_case
```

(RTL simulator ignores, but DC understands)

```verilog
reg A, B;
wire [2:0] SEL ;
always@(*)
{A, B} = 2’b0 ;
casez(SEL) // synopsys_parallel_case
  3’b?11: B = 1 ;
  3’b1??: A = 1 ;
endcase
```

Put it after the case declaration

11.13

11.14
Case Directives Supported by DC (2)

- **Synopsys_full_case**
  - Treats o/p of unstated case items as **don’t care**
  - Ignored if the case statement is complete
  - As a result, **no latch** is created

- **Synopsys_parallel_case**
  - Treats all case items as **non-overlapped**;
    implement each of them separately
  - As a result, **no priority logic** is created

**Note:** Potential discrepancy b/w RTL simulation and gate-level simulation

Example: RTL the Same, Gate-Level Different

```verilog
reg A, B;
wire [2:0] SEL;
always@(*)
{A, B} = 2'b0;
casez(SEL)
  3'b?11: B = 1;
  3'b1??: A = 1;
endcase
```

```verilog
reg A, B;
wire [2:0] SEL;
always@(*) // synopsys_parallel_case
{A, B} = 2'b0;
casez(SEL)
  3'b?11: B = 1;
  3'b1??: A = 1;
endcase
```

Lec. 8.46

Priority Logic

Independent Logic
Logic Synthesis

- Conceptual overview & tool setup
- RTL modification

**Technology libraries**

- Design environment & constraints
- Major synthesis commands
- Gate-level simulation

What are Technology Libraries About? (1)

```plaintext
library ("saed32rvt_ss0p95v125c") {
  operating_conditions (ss0p95v125c) {
    process : 0.99;
    voltage : 0.950000;
    temperature : 125.000000;
    tree_type : "best_case_tree";
  }
  wire_load (ForQA) {
    capacitance : 0.029396;
    resistance : 2.273000e-03;
    area : 0.010000;
    slope : 30.285426;
    fanout_length("1", "8.2750360");
    ...
  }
  wire_load ("8000") {
    ...
  }
  ...
  wire_load_selection (predcaps) {
    wire_load_from_area(0.000000,200.000000, "ForQA");
    wire_load_from_area(200.000000,8000.000000, "8000");
    ...
  }
}
```

**Operating condition**
(process, voltage, temp.)

**Wire-load model**
(rough est. of wiring parasitics before PnR)

**Wire-load selection**
(based on synth. area of your design)
What are Technology Libraries About? (2)

- Detailed information of each standard cell

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>Area</th>
<th>Functionality</th>
<th>Pin Capacitance</th>
<th>Leakage Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>7x7 Look-up Tables (LUT) for Dynamic Power &amp; Propagation Delay of all i/o Paths (idx1: input transition time; idx2: o/p capacitance)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What are Technology Libraries About? (3)

```plaintext
cell (AND2X1_RVT) {
  area : 2.033152; cell_leakage_power : 1.404953e+05;
  leakage_power () { when : "!A1&&!A2"; value : "5.6919234e+04"; }
  ... pin (Y) {
    related_power_pin : "VDD"; related_ground_pin : "VSS"; direction : "output";
    ... max_capacitance : 8.000000; max_transition : 0.194746;
    internal_power () {
      related_pin : "A1";
      rise_power ("power_outputs_1") {
        index_1("0.016, 0.032, 0.064, 0.128, 0.256, 0.512, 1.024");
        index_2("0.1, 0.25, 0.5, 1, 2, 4, 8");
        values("0.9612064, 0.9624440, 0.9509065, 0.9587265, 0.9569466, 0.9523558, 0.9245481", \ 
          ... )
      }
      timing () {
        related_pin : "A1";
        cell_rise ("del_1_7_7") {
          index_1("0.016, 0.032, 0.064, 0.128, 0.256, 0.512, 1.024");
          index_2("0.1, 0.25, 0.5, 1, 2, 4, 8");
          values("0.0400985, 0.0415339, 0.0437422, 0.0477826, 0.0546696, 0.0667816, 0.0888995", \ 
            ... )
        }
      }
  }
```

11.19

11.20
Access Synopsys 32nm Technology Libs.

Path: /w/apps2/public.2/tech/synopsys/32-28nm/SAED32_EDK/lib/stdcell_rvt/db_nldm
(rvt: regular-V_{th}. Can change to hvt and lvt for high-V_{th} and low-V_{th} libraries)

There you can find various libraries, each with different design corners. We select saed32rvt_ff1p16vn40c.db for hold-time analysis & saed32rvt_ss0p95v125c.db for setup-time analysis

Note: .db files are only used by DC. If you want to read library information as in page 11.18~20, access the corresponding .lib files

Setup Technology Libraries

set search_path "\$search_path \\
/w/apps2/public.2/tech/synopsys/32-28nm/SAED32_EDK/lib/stdcell_rvt/db_nldm"
set target_library "saed32rvt_ff1p16vn40c.db saed32rvt_ss0p95v125c.db"
set link_library "* saed32rvt_ff1p16vn40c.db saed32rvt_ss0p95v125c.db dw_foundation.sldb"
set synthetic_library "dw_foundation.sldb"

# Define work path (note: The work path must exist, so you need to create a folder WORK first)
define_design_lib WORK -path ./WORK
set alib_library_analysis_path "./alib-52/"

(set above scripts in a .tcl file, and source it after you open DC)

- DC follows search_path to find libraries you specify
- DC uses cells in target library for logic optimization (so we need to do set_target_library first)
- Remember: Use backslash \ before changing lines to avoid compilation errors.
Logic Synthesis

- Conceptual overview & tool setup
- RTL modification
- Technology libraries

Design environment & constraints
- Major synthesis commands
- Gate-level simulation

Synthesis is Constraint-Driven

You set the goals (through design constraints)
DC optimizes the design to meet your goals
Logic Synthesis is Timing-Driven

your current design

- **Input delay**: Arrival of an external path w.r.t. the clock edge
- **Output delay**: timing from an o/p (of current design) to a register i/p (of other submodules)

Describe Design Constraints

- **Clocks**
  - Period, latency, uncertainty
- **Design rules**
  - Maximum transition
  - Maximum capacitance
  - Maximum fanout
- **Input-related**
  - Driving cells, Input delay
- **Output-related**
  - Load, Output delay
- **Exception paths**
  - False paths, multi-cycle paths
- **Optimization goal**
  - Maximum area/power

# Especially important for bottom-up design methodology

# Accurate constraints (not too tight/loose): Good integ. results
Ideal vs. Real Clock

![Ideal vs. Real Clock diagram]

- **Ideal clock**
- **Period**
- **Latency**
- **Uncertainty** (jitter + skew)
- **Transition**

**Courtesy:** Synopsys

Clock Description

- **create_clock**: define clock’s waveform (e.g. period)
  
  ```
  create_clock -name "CLK" -period 2.0 [get_ports "CLK"]
  ```

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

- **set_dont_touch_network**: do not buffer clock network
  
  ```
  set_dont_touch_network [get_clocks "CLK"]
  ```

- **Specify uncertainty (skew + jitter) of clock network**
  
  ```
  set_clock_uncertainty 0.1 [get_clocks "CLK"]
  ```
**Maximum Transition**

Sets the `max_transition` attribute to a specified value on specified clocks group, ports or designs.

```
set_max_transition 0.15 current_design
```

**Tip:** Check the `max_transition` variable in `.lib` file to setup more realistic numbers.

---

**Maximum Capacitance**

Sets the `max_capacitance` attribute to a specified value on the specified input ports and designs.

```
set_max_capacitance 1.000 [get_designs *]
```

**Tip:** Check the `max_capacitance` variable in `.lib` file to setup more realistic numbers.

("*" means “all designs including submodules”)
Maximum Fanout

Sets the `max_fanout` attribute to a specified value on specified input ports and/or designs.

```
set_max_fanout 1.000 [get_designs *]
```

Tip: Check the `fanout_load` variable of cells in `.lib` file to setup more realistic numbers.

---

Output Load

Sets the `set_load` attribute to a specified value on specified output ports and/or designs.

```
set_load 1.000 [all_outputs]
```

Tip: Check the `capacitance` variable of cell input pins in `.lib` file to setup more realistic numbers.
Driving Cells

Sets attributes on input or inout ports of the current design, specifying that a library cell or output pin of a library cell drives specified ports.

```
set_driving_cell [-library lib] [-lib_cell lib_cell_name] [-pin pin_name]
```

```
set ALL_IN_BUT_CLK [remove_from_collection [all_inputs] “CLK”]
set_driving_cell -no_design_rule -library saed32rvt_ss0p95v125c.db:saed32rvt_ss0p95v125c -lib_cell DFFASRX2_RVT -pin Q $ALL_IN_BUT_CLK
```

Input & Output Delay

Sets delay on pins or ports relative to a (ideal) clock signal.

```
set_input_delay -max delay_value [-clock clock_name]
set_input_delay -min delay_value [-clock clock_name]
set_output_delay -max delay_value [-clock clock_name]
set_output_delay -min delay_value [-clock clock_name]
```

```
set_input_delay 0.6 -clock “CLK” $ALL_IN_BUT_CLK
set_input_delay -min 0.3 -clock “CLK” $ALL_IN_BUT_CLK
set_output_delay 0.6 -clock “CLK” [all_outputs]
set_output_delay -min 0.3 -clock “CLK” [all_outputs]
```
**False Paths**

Removes timing constraints from particular paths, but still needs to satisfy design rule (transition, capacitance, fanout).

```plaintext
set_false_path -from [from_list]
```

```plaintext
set_false_path -from [get_ports ACLR_]
set_false_path -from [get_ports SI]
```

**Maximum Area/Power**

Optimization goals for your design (DC will do it best to satisfy them, w/o violating the three design rules)

```plaintext
set_max_area 0.0
```

desired area

```plaintext
set_max_total_power 0.0
```

desired power
Operating Condition

Defines the operating conditions for the current design.

```
set_operating_conditions
[-min min_condition] [-max max_condition]
```

```shell
set_operating_conditions
-min ff1p16vn40c -max ss0p95v125c
```

for hold-time check    for setup-time check

Wire-load Models and Modes

Specify a selection group to use for determining a wire load model to be assigned to designs and cells or to a specified cluster.

```
set_wire_load_selection_group group_name -max
set_wire_load_selection_group group_name -min
```

```shell
set_wire_load_selection_group "predcaps"
```

DC will select proper model based on synthesis area (again, lookup this name in .lib file)
Logic Synthesis

- Conceptual overview & tool setup
- RTL modification
- Technology libraries
- Design environment & constraints

† Major synthesis commands
- Gate-level simulation

Preview: Use Synthesis Commands Properly

- Not all commands introduced are necessarily needed
- Things MUST do
  - Analyze, elaborate, and link design
  - Compile design
- Things can do based on your needs
  - Ungroup; Uniquify
  - Clock gating creation
  - Remove unconnected ports

**Suggestion**: Learn by playing with different commands and observing the differences
Analyze Designs

Analyzes the HDL files and stores the intermediate format in the specified library.

analyze [-format vhdl | verilog | sverilog] file_list

analyze –format verilog {adder.v}

Elaborate & Link Designs

Builds a design from the intermediate format of a Verilog module, a VHDL entity and architecture, or a VHDL configuration.

elaborate design_name

set DESIGN_NAME adder
elaborate $DESIGN_NAME
### Ungroup & Uniquify

Removes a level of hierarchy.

\[ \text{ungroup cell\_list | -all [-flatten]} \]

**ungroup -flatten -all**

Removes multiple-instantiated hierarchy in the current design by creating a unique design for each of the cell instances.

\[ \text{uniquify [-force] [-cell cell\_list]} \]

**uniquify**

---

### Compile

Performs logic-level and gate-level synthesis and optimization on the current design.

\[ \text{compile [-no\_design\_rule | -only\_design\_rule | -only\_hold\_time] [-map\_effort medium | high] [-boundary\_optimization]} \]

**compile –only_design_rule**

**compile –map_medium high**

**compile –boundary_optimization**

**compile –only_hold_time**
Gated Clock Creation

Four commands to construct clock gating
(note: Modify RTL as in pp. 13 to enable this feature)

```bash
set_clock_gating_registers -include_instances [all_registers -clock "CLK"]
set_clock_gating_style -num_stages 2 -sequential_cell latch -minimum_bitwidth 8 -max_fanout 32
insert_clock_gating -global
propagate_constraints -gate_clock
```

Remove Unconnected Ports

Removes unconnected ports or pins from cells, references, and subdesigns.

```bash
remove_unconnected_ports -blast_buses [get_cells -hier]
```
Report Designs

1. `report_timing` -path full -delay min -max_paths 10 -nworst 2 > `Design.holdtiming`
2. `report_timing` -path full -delay max -max_paths 10 -nworst 2 > `Design.setuptiming`
3. `report_area` -hierarchy > `Design.area`
4. `report_power` -hier -hier_level 2 > `Design.power`
5. `report_resources` > `Design.resources`
6. `report_constraint` -verbose > `Design.constraint`
7. `check_design` > `Design.check_design`
8. `check_timing` > `Design.check_timing`

Logic Synthesis

- Conceptual overview & tool setup
- RTL modification
- Technology libraries
- Design environment & constraints
- Major synthesis commands

⚠️ Gate-level simulation
Slight Modification of Original Testbench.v

- RTL simulation: `design.v + testbench.v`
- Gate-level simulation: `design.vg + design.sdf` (from DC) + `testbench.v` (w/ slight modification)

```verilog
`timescale 1ns/10ps
module Test;
ADD AddSim(…);
...
...
...
...
endmodule
```

```verilog
`timescale 1ns/10ps
module Test;
ADD AddSim(…);
initial begin
	$sdf_annotate("design.sdf", AddSim);
end
...
...
endmodule
```

Any Questions?

The “Man” is always there for you, 24/7 😊
(type `man <syntax you want to lookup>` in DC)

or check out various user manuals online
## Synthesis Summary

- **Synthesis tool makes VLSI design a lot easier**
  - Easy to use: `.v` files + `.tcl` files
  - Easy to switch designs to any technology by changing associated libraries
  - Easy to have accurate area/timing/power estimate
  - Easy to match the best architecture with design constraints

- **But... not a substitute for thinking**
  - Mind your coding styles
  - Handmade optimization (e.g. using shift-and-add for constant multiplication) still needed