CSCE 3953 System Synthesis and Modeling
Lecture 6 Synthesis with Synopsys Design Compiler

Instructor: Dr. Jia Di

Some slides are borrowed from Synopsys Galaxy 2006 Seminar Series
Outline

- Synthesis Overview
- Design Compiler Flow
  - Design Compiler Setup
  - Reading the design
  - Design Constraints
  - Compile Strategies
  - Design Analysis
Synthesis

Combining pre-existing elements to form something new
Logic Synthesis

Combining primitive logic functions to form a design netlist that meets functional and design goals

Functional Description (HDL)
Goals

Netlist
Why Synthesis?

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

# of Elements  
Complexity  

Effort  
Time  
Cost  

Layout
Logic Synthesis

Logic Synthesis = Translation + Mapping + Optimization

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

Hardware Description Language (HDL)

Translation (read)

Generic Boolean (GTECH)

Mapping/Optimization (compile)

Target Technology (standard cells)
Functional Description

- Written in Hardware Description Language (HDL)
  - Verilog/VHDL
  - Register Transfer Level (RTL)
    - Synchronous => reliable behavior
    - Simplifies timing verification
    - Simplifies optimization algorithms
    - Optimal results
- Coding style affects results
Translation

- Converts HDL to functional boolean equivalent
  - HDL syntax/rule checks
  - Optimizes HDL
  - Arithmetic function mapping
  - Sequential function mapping
  - Combinational function mapping

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

- Maps Boolean functions to technology specific primitive functions
- Modifies mapping to meet design goals
  - Design Rules
  - Timing
  - Area
  - Power

Generic Boolean (GTECH)

Mapping/Optimization (compile)

Target Technology (standard cells)
Design goals (constraints) drive optimization
STA breaks designs into sets of signal paths

Each path has a startpoint and an endpoint:

- **Startpoints**:
  - Input ports
  - Clock pins of Flip-Flops or registers

- **Endpoints**:
  - Output ports
  - All input pins of sequential devices (except clock pins)
Optimization: Slack-Driven

FF1/clk

FF2/D

FF2/clk

Data Arrival

Data Required

Clk

Data Arrival

Data Required

Setup

1.1ns

5.1ns

1ns

5ns
residue = 16'h0000;
if (high_bits == 2'b10)
  residue = state_table[index];
else
  state_table[index] = 16'h0000;

Hardware Description Language (HDL)

Target Technology (standard cells)
Outline

- Synthesis Overview
- Design Compiler Flow
  - Design Compiler Setup
  - Reading the design
  - Design Constraints
  - Compile Strategies
  - Design Analysis
Invoking DC

> dc_shell -tcl

or

> design_vision -tcl

setenv PATH $SYNOPSYS/$ARCH/syn/bin/dc_shell

- $SYNOPSYS - installation location on your network
- $ARCH - linux, sparcOS5, sparc64, etc...

Best Practice:

- Early in the design phase is a good time to decide on design naming conventions, design style guides, common design directory structures, and revision control systems.
DC Setup Files

- Setup files automatically read at DC startup
  - .synopsys_dc.setup
- Possible locations (read in this order)
  1. Root setup:
     - $SYNOPSYS/admin/setup/.synopsys_dc.setup
  2. Home setup:
     - $HOME/.synopsys_dc.setup (optional)
  3. Local setup:
     - ~/.synopsys_dc.setup (optional)
- Use to customize the work environment
# TCL-subset .synopsys_dc.setup file must
# have the # character on the first line of the file

set search_path ". /synopsys/libraries/syn $search_path"
set target_library "lsi_10k.db"
set synthetic_library "standard.sldb dw_foundation.sldb"
set link_library "* $target_library $synthetic_library"
set symbol_library "lsi_10k.sdb"

define_design_lib MY_WORK -path ./WORK

# example: removing high drive inverter
set_dont_use lsi_10k/IVP
Library Setup

- **search_path**
  - Allows files to be read in without specifying directory path in the command
  - Directories in which DC will look for library/design .db files during a link

- **target_library**
  - Technology cell library files (e.g., lsi_10k.db)
  - Compile chooses inferred cells from target library
Library Setup - continued

- **synthetic_library**
  - Library of DesignWare components
  - **dw_foundation.sldb**
    - Advanced set of IP components optional to DC
    - Wide variety moderate/high performance arithmetic architectures
    - Fifos, stacks, counters, digital PLL, arbiters, priority encoders, SRAM models, ECC, CRC, debugger, decoders/encoders, more...
    - Macrocells: 8051 microcontroller, 16550 UART, Memory BIST controller, AMBA peripherals(I2C, UART, SSI, APB, AHB, …)
• **link_library**
  - Used during design linking (pre- and post-compile)
  - All cells in a design must be in one of the link libraries
    - Inferred (chosen during compile based on RTL functionality)
    - Instantiated (specific cell instance placed in design RTL)
  - **link_library** must always start with "*", indicating loaded designs should be searched first when linking
  - All synthetic and target libraries must be included in **link_library**
Library Setup - continued

- define_design_lib
  - Directory where DC places intermediate design files (default is directory in which DC is run)
- set_dont_use <lib>/<cell>
  - Specifies cells of a target library or implementations of a synthetic library to not use during compile

**Best Practice:**
- If your technology library has many drive strengths for each cell function, consider using `set_dont_use` for the highest drive strength of each cell function. After routing, if you need to up-size cells to a higher drive to overcome larger than anticipated capacitances, you can remove the `set_dont_use`.
Outline

- Synthesis Overview
- Design Compiler Flow
  - Design Compiler Setup
  - Reading the design
  - Design Constraints
  - Compile Strategies
  - Design Analysis
Analyze

- Translates HDL to intermediate format
- Recommended for reading RTL

```
dc_shell-t> analyze -help
Usage: analyze    # analyze
   [-library library_name]  (Use this library as the work library)
   [-work library_name]    (Use this library as the work library)
   -format format string   (The format of the hdl files)
   [-update]               (Update analysis from original source)
   [-schedule]             (Analyze the design for scheduling)
   [-create_update]        (Create .update file for use by
                           "analyze -update")
   [-define macro_names]   (list of top-level macros, Verilog only)
   file_list               (Files to read in)
```
Elaborate

- Second step of HDL translation
- Builds generic technology (GTECH) database
  - HDL parameters are expanded
  - Registers and latches are inferred
  - Links design
- Supports parameter passing/architecture selection
- Recommended for reading RTL

```
dc_shell-t> elaborate -help
Usage: elaborate  # elaborate
   [-library library_name] (Use this library as the work library)
   [-work library_name]  (Use this library as the work library)
   [-architecture arch_name] (Architecture to build)
   [-parameters param_list] (Parameters for the design)
   [-file_parameters file_list] (Files containing parameters for the design)
   [-update] (Automatically update out-dated files)
   [-schedule] (Build the design for scheduling)
   [-gate_clock] (Gate clocks)
   design name (Name of the design to build)
```
read_file

- read_file performs analysis and elaboration (except link) in one step
- Parameter passing/architecture selection not supported
- Recommended for reading mapped netlists

```
dc_shell-t> read_file -help
Usage: read_file  # read file from disk
    [-format format_name]  (verilog, vhdl, ddc, db)
    [-single_file file_name]
                        (group all designs into this file)
    [-define macro_names]  (list of top-level macros, Verilog and SystemVerilog only)
    [-library library_name]
                        (Use this library as the work library, VHDL only)
    [-work library_name]   (Use this library as the work library, VHDL only)
    [-names_file file_list]
                        (list of files for name changes)
    [-ilm]                (Read from the Milkyway ILM view)
    [-rtl]                (register transfer-level verilog/vhdl format)
    file_list             (list of files to read)
```
Outline

- Synthesis Overview
- Design Compiler Flow
  - Design Compiler Setup
  - Reading the design
  - Design Constraints
  - Compile Strategies
  - Design Analysis
Design goals \textit{(constraints)} drive optimization
Two types of constraints
  - Design Rule Constraints (DRC)
  - Optimization Constraints

DC calculates cost functions for each type
Optimization attempts to minimize cost functions

Beginning Mapping Optimizations (Medium effort)

<table>
<thead>
<tr>
<th>ELAPSED TIME</th>
<th>AREA</th>
<th>WORST NEG SLACK</th>
<th>TOTAL NEG SLACK</th>
<th>DESIGN RULE COST</th>
<th>ENDPOINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:34</td>
<td>9992.8</td>
<td>9.47</td>
<td></td>
<td>219.1</td>
<td>25.1</td>
</tr>
<tr>
<td>0:00:35</td>
<td>6896.3</td>
<td>9.48</td>
<td></td>
<td>218.2</td>
<td>25.1</td>
</tr>
<tr>
<td>0:00:35</td>
<td>7001.9</td>
<td>9.48</td>
<td></td>
<td>217.6</td>
<td>25.1</td>
</tr>
<tr>
<td>0:00:36</td>
<td>7000.8</td>
<td>9.48</td>
<td></td>
<td>217.4</td>
<td>25.1</td>
</tr>
</tbody>
</table>
Optimization Priority

- Design goals often conflict
- Optimization engines must resolve conflict
- Priority rules
- DC priority
  - DRC
  - Timing
  - Power
  - Area
- Can be modified with `set_cost_priority`
Constraint Guidelines

- **Golden rule #1:** set realistic constraints
  - Most critical for timing and DRC constraints
- **Golden rule #2:** validate constraints
  - check_design
  - check_timing
  - report_timing_requirements

- **Correlation issues**
  - Derate
    - Adjusting constraints to account for unmodeled effects
  - Over-constraining
    - Modifying constraints to drive optimization to desired goal
Constraint Validation

- **check_design**
  - Checks internal DC representation for design consistency
  - Run before compile

- **check_timing**
  - Verifies timing setup is complete
  - Run before and after compile

- **report_timing_requirements**
  - Reports design database constraint attributes
  - Used to debug timing exceptions
  - Use `-expanded` to report all valid exceptions
  - Use `-ignored` to report invalid exceptions

**Best Practice:**
- Review the output of `check_design` and `check_timing` very closely. Make sure every Error is fixed and every Warning is fully understood (and preferably fixed).
Design Rule Constraints (DRC)

- `set_max_transition`
  - Largest transition time allowed
- `set_max_fanout`
  - Largest fanout allowed
- `set_max[min]_capacitance`
  - Largest/smallest capacitance allowed
- Defaults usually set in technology library
- Highest optimization priority
Optimization Constraints: Operating Conditions

- `set_operating_conditions`
  - Sets PVT for timing calculations
- Process/Voltage/Temperature (PVT) conditions affect timing
- Technology libraries are characterized at different PVT corners
- Corner specified in technology library (i.e., worst, typical, best)
Optimization Constraints: Net Parasitics

- DC Topographical (DCT) calculates net parasitics based on physical layout
  - Correlates well to place and route timing
- Wireload Models (WLM)
  - set_wireload_model, set_wireload_mode
  - Fanout-based statistical model for estimating wire capacitance
  - Needed for non-DCT runs
  - Not as accurate as DCT method
  - Default wireloads provided in technology library generally inaccurate
  - Custom wireload generated from accurate floorplan of the design gives best results
Optimization Constraints: Input/Output

- Inputs
  - `set_input_delay`
    - Models delay from signal source to design input port
  - `set_driving_cell -input_transition_rise[fall]`
    - Models input signal slew
    - Alternatively `set_input_transition`
      - Not as accurate

- Outputs
  - `set_output_delay`
    - Models delay from design output port to signal destination
  - `set_load`
    - Models load on output port
set_operating_conditions -lib lsi_10k WCCOM
set_wire_load_mode top
set_wire_load_model -name blockAw1 [get_design L1a]

set_driving_cell -lib_cell IVA -lib lsi_10k \
  -input_transition_rise 0.4 \ 
  -input_transition_fall 0.5 \ 
  -from_pin A -pin Z [get_ports data_in*]

set_load [load_of lsi_10k/IVA/A] [get_ports data_out*]

set_input_delay -clock CK 10 [get_ports data_in*]
set_output_delay -clock CK 5 [get_ports data_out*]
Optimization Constraints: Clocks

- `create_clock`
  - Name, source port, period, duty cycle
  - Constrains timing on all register to register paths
- `set_clock_uncertainty`
  - Estimated network skew

**Best Practice:**
- DC will run more efficiently with clocks that have a common base period that is small. For example, 2 clocks with periods 20ns and 30ns have a common base period of 60ns - this will work well in DC. However, for periods of 10ns and 10.1ns, the common base period is 1010ns! By changing the 10.1ns clock to a 10ns clock, with an uncertainty of 0.1ns, timing analysis results will be equivalent, and DC will run much more efficiently.
Optimization Constraints: Clocks

- `set_clock_latency`
  - Estimated source and network delays
- `set_clock_transition`
  - Estimated input slew
- `set_ideal_network`
  - Disables timing calculation and optimization of clock network
  - On by default
Optimization Constraints: Clocks

- Internally generated clocks
  - create_generated_clock
    - Calculates latency from source port to internal pin
- Post-CTS
  - set_propagated_clock
    - Calculates actual clock tree delays
  - set_dont_touch_network
    - Prevents clock tree optimization
Optimization Constraints: Clocks

- **Virtual clock**
  - Create with `create_clock` with no clock source port
  - Useful for modeling external clocks
  - Used for input/output delay specification

- **Gated clocks**
  - `set_clock_gating_checks`
    - Delay constraint to prevent clock glitching
Example

pre-layout

```bash
create_clock -p 30 -n MCLK Clk
set_clock_uncertainty 0.5 MCLK
set_clock_transition 0.25 MCLK
set_clock_latency -source 4 MCLK
set_clock_latency 2 MCLK
```

post-layout

```bash
create_clock -p 30 -n MCLK Clk
set_clock_uncertainty 0.1 MCLK
set_clock_transition 0.25 MCLK
set_clock_latency -source 4 MCLK
set_propagated_clock MCLK
```
Optimization Constraints: Derating

- `set_timing_derate`
  - Mechanism to add margin
  - Used to account for unmodeled delay affects
  - Used to adjust for correlation problems
Optimization Constraints: Power

- `set_max_dynamic_power`
- `set_max_leakage_power`
- `set_max_total_power`

- Power constraints are lower priority than timing constraints
  - Optimization engines will not reduce power if it creates negative slack
Optimization Constraints: Area

- `set_max_area`
  - Constrains design area
  - Area constraints are lower priority than timing constraints
    - Optimization engines will not reduce area if it creates negative slack
Optimization Constraints: Structural

- **set_fix_multiple_port_nets**
  - Buffer all net segments connected to an output port

- **set_dont_touch**
  - Prevents optimization
  - Works only for mapped gates

- **set_dont_touch_network**
  - Same as **set_dont_touch**
  - Applies to all combinational logic in fanout

- **set_size_only**
  - Prevents all optimizations except sizing
Port function

- set_logic_one
  - Assigns logic one state to port
- set_logic_zero
  - Assigns logic zero state to port
- set_logic_dc
  - Assigns don’t care state to port
- set_equal
  - Defines two ports to have equivalent logic states
- set_opposite
  - Defines two ports to have opposite logic states
Design For Test (DFT)

- `set_dft_configuration`
  - Setup DFT adaptive scan insertion, violation fixing, observability improvement options
- `set_dft_insertion_configuration`
  - Setup scan insertion options
- `set_scan_configuration`
  - Setup scan insertion options
- `set_dft_drc_configuration`
  - Setup DFT DRC checking options
- `set_dft_signal`
  - Scan chain configuration
- `set_scan_element`
  - Scan chain configuration
group_path

- Group paths into a separate optimization unit
- Worst violator in every path group is fully optimized
- By default, groups are created for each clock domain
- DC optimizes each path group individually
set_critical_range

- Default algorithm optimizes only worst violator in each path group, i.e., critical range = 0
- Critical range > 0
  - Algorithm will optimize all paths within critical range of worst violator in each path group
  - More paths are optimized
  - Total negative slack (TNS) is reduced
  - Increases runtime so use cautiously

set_cost_priority

- Modifies optimization priority of constraints
set_max_area 0
set_fix_multiple_port_nets -all [all_designs]
# recommended to separate I/O paths from reg-reg paths
group_path -name INPUT -from [all_inputs] -to [all_clocks]
group_path -name OUTPUT -to [all_outputs] -to [all_clocks]
group_path -name COMBO -from [all_inputs] -to [all_outputs]
set_cost_priority -delay
set_critical_range 0.2 top_design
set_timing_derate -max -early 0.95
Set_timing_derate -max -late 1.1
set_max_leakage_power 5 mW

Best Practice:
"report_timing" output shows each path group separately, so make sure to review each timing report completely. Consider using "report_constraint -all_violators" to show all violators from all path groups (file can be large!).
Outline

- Synthesis Overview
- Design Compiler Flow
  - Design Compiler Setup
  - Reading the design
  - Design Constraints
  - Compile Strategies
  - Design Analysis
Definitions

- **Top-down Methodology**
  - Compiling only at the top level

- **Bottom-up Methodology**
  - Compiling each lower level module separately
  - Link all the modules together for top level integration

Which methodology to use is design dependent. Choose the one that will work best with your designs.
Top-Down Compile

Benefits:
- Optimization engines work on full design, complete paths
- Usually get best optimization result
- No iteration required
- Simpler constraints
- Simpler data management

Drawbacks:
- Longer runtime
  - More processing required
  - More memory required

```
dc_shell-t> current_design TOP
dc_shell-t> compile_ultra –timing_high_effort_script
```
Bottom-Up Compile

**Benefits:**
- Divide-and-conquer methodology localizes problem areas
- Budgeting method enables parallelized synthesis effort
- Less processing required per run
- Less memory required per run

**Drawbacks:**
- Optimization works on sub-designs, hierarchical path segments
- Optimization result not as good as top-down compile
- Iterations may be required
- More hierarchies and data to maintain
- More work for user
- More error-prone

**Best Practice:**
- Perform a `compile -top` to touch up the critical path and DRC at the top level.

*Note:* `compile -top` fixes DRC and top-level timing violations. It only applies to paths crossing top-level hierarchical boundaries.
Guidelines

- Always try top-down first
- Best compile methodology to use is design dependent
- Trading off quality of results for runtime
Meeting Timing Goals

- Enable DC-Ultra optimizations and embedded script
  - `compile_ultra -timing_high_effort_script`
- If runtime excessive
  - `compile_ultra`
    - `compile -map_effort_high -incr`
- Use critical range and path groups
- Ungroup hierarchy
- Register retiming
- Use fast DesignWare
• Advanced datapath synthesis
  • Optimized arithmetic trees
  • Carry save logic
• Embedded scripts targeting area/timing
• Automatic ungrouping
• Automatic boundary optimization
• Topographical technology for best timing correlation
• Library aware structuring and mapping
Critical range > 0 reduces TNS and improves overall timing
- `set_critical_range 2 TOP`

Path groups focus optimization effort

Create additional path groups and set a critical range
- `group_path -from [all_inputs] -name input_paths -critical_range 0`
- `group_path -to [all_outputs] -name output_paths -critical_range 0`
- `group_path -from clk -to clk -name internal_paths -critical_range 2`

Group critical paths
- `group_path -to top/mem*/data -name memory_inputs -critical_range 2`

**Best Practice:**
- Use a reasonable amount for critical range (i.e. 10% of the clock period). The larger the critical range, the longer the compile time.
Optimization works on paths within hierarchical boundaries.
Removing hierarchical boundaries allows optimization algorithms to work on larger or entire paths.
Produces better optimization results.
For best timing results, remove hierarchy on critical paths.
For best area results, remove as much hierarchy as possible.
`compile_ultra` automatically ungroups as needed.
`ungroup` allows manual ungrouping.
Register Retiming

- Moves registers through combinational gates to improve timing/area
  - optimize_registers/set_optimize_registers
    - Use for pipelined designs and aggressive retiming
  - pipeline_design
    - Use to pipeline designs
  - compile_ultra -retime
    - Use for non-pipelined designs and less aggressive retiming
Use Fast DesignWare Components

- `set_dont_use` on slow DesignWare components
  - `set_dont_use standard.sldb/DW01_add/rpl`
Area Consideration

- Enable DC-Ultra optimizations and embedded script
  - compileUltra -area_high_effort_script
- If not using -area_high_effort_script set max area constraint
  - set_max_area 0
  - -ignore_tns only if design easily meets timing
- Clock gating
- Ungroup hierarchy
- Register retiming
- Modify DesignWare selection settings

**Best Practice:**
- Try turning on one switch at a time. There will be a significant impact to runtime if all switches are turned on for one compile.
Clock Gating

- Clock gating not only can save power, it can also help in area savings
  - insert_clock_gating
  - Run before compile_ultra

saves one mux per register, by using only one clock gate per register bank
- Enable only ripple adder (rpl) implementation of DW
  - `set_implementation DW01_add/rpl A1`
- Change High Level Optimization (HLO) variable settings
  - `set hlo_resource_implementation area_only`
  - `set hlo_resource_allocation area_only`
  - `set hlo_share_common_subexpressions true`
Runtime Methodology

- Use low / medium effort `compile`
- `set_dont_use certain implementations of DW`
  - minimize the choices during compile
- Watch for asynchronous clock domains
  - should have a low common base period
- Specify timing exceptions efficiently with wildcard
Summary

- Ensure constraints are clean, efficient, and realistic
- Use `compile_ultra`
- Use optimization strategy targeted for your design goals
- Start with a minimal set of variables/commands for the 1st compile, then add one switch at a time for subsequent compiles
Outline

- Synthesis Overview
- Design Compiler Flow
  - Design Compiler Setup
  - Reading the design
  - Design Constraints
  - Compile Strategies
  - Design Analysis
Analysis Flow

- **Before optimization**
  - Verify constraints and attributes
  - Check design consistency

- **During optimization**
  - Customize compile log
  - Checkpoint compile run

- **After optimization**
  - Verify timing and DRC constraints are satisfied
  - Refer to Design Compiler User Guide for debug scenarios and more commands
- Operating conditions
- Wire load model and mode
- Internal input and output pin delays
- Disabled timing arcs
- Clock definition
- Clock latency
- Clock skew
Use `check_design` to check consistency

Consistency means …
- no unintentionally unconnected ports
- no unintentionally tied ports
- no cells without input or output pins
- no mismatch between a cell and its reference
- no multiple-driver nets
- no recursive hierarchy

Best Practice:
Always ensure consistency before proceeding to synthesis -- although inconsistencies that affect functionality are often caught before synthesis during simulation, sometimes a design or block is not completely simulated before synthesis … legacy designs, design exploration, etc.
- **Unconstrained timing paths**
  "Warning: The following end-points are not constrained for maximum delay."

- **Clock-gating logic**
  "Warning: The clock network starting at 'clk' is gated by the following input pins. The gating timing arcs might need to be disabled for clocks with the 'propagated_clock' attribute."

- **Unmapped cells**
  "Warning: Design 'design_name' contains unmapped cells."
Customizing Compile Log

- Printed during “Trials” phase of optimization
- Default fields
  - elap_time, area, wns, tns, drc, endpoint
- Additional fields
  - group_path, max_delay, min_delay, mem, time, trials, cpu, dynamic_power, leakage_power
- Use `compile_log_format` variable to customize
  - `set compile_log_format \ "%elap_time %mem %wns %group_path %endpoint"`
- Timing summary for all path groups
- Good overall status of design timing
Shows difference between user constraints and actual design values

dc_shell> report_constraint
************************************************
Report : constraint
Design : counter
Version: 2000.11
Date : Fri Jun 15 15:49:46 2001
************************************************
Weighted
Group (max_delay/setup) Cost Weight Cost
------------------------------------------------
CLK 0.00 1.00  0.00
default 0.00 1.00  0.00
------------------------------------------------
max_delay/setup 0.00
Constraint Cost
------------------------------------------------
max_transition 0.00 (MET)
max_fanout 0.00 (MET)
max_delay/setup 0.00 (MET)
critical_range 0.00 (MET)
min_delay/hold 0.40 (VIOLATED)
max_leakage_power 6.00 (VIOLATED)
max_dynamic_power 14.03 (VIOLATED)
max_area 48.00 (VIOLATED)
min_porosity 2.00 (VIOLATED)
**Report of all timing/drc violations**

**Best Practice:**

*Use* `report_timing -nworst` *for multiple timing violations per path endpoint* … *`report_constraint` reports only one path per endpoint*
### Path timing report

- Extremely flexible
  - -from, -to, -through,
  - path, -delay,
  - nworst, -
  - max_paths,
  - input_pins, -nets,
  - transition_time,
  - capacitance,
  - attributes,
  - physical,

---

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>input external delay</td>
<td>0.00 0.00 f</td>
</tr>
<tr>
<td>cin (in)</td>
<td>0.00 0.00 f</td>
</tr>
<tr>
<td>U19/Z (AN2)</td>
<td>0.87 0.87 f</td>
</tr>
<tr>
<td>U18/Z (EO)</td>
<td>1.13 2.00 f</td>
</tr>
<tr>
<td>add_8/U1_1/CO (FA1A)</td>
<td>2.27 4.27 f</td>
</tr>
<tr>
<td>add_8/U1_2/CO (FA1A)</td>
<td>1.17 5.45 f</td>
</tr>
<tr>
<td>add_8/U1_3/CO (FA1A)</td>
<td>1.17 6.62 f</td>
</tr>
<tr>
<td>add_8/U1_4/CO (FA1A)</td>
<td>1.17 7.80 f</td>
</tr>
<tr>
<td>add_8/U1_5/CO (FA1A)</td>
<td>1.17 8.97 f</td>
</tr>
<tr>
<td>add_8/U1_6/CO (FA1A)</td>
<td>1.17 10.14 f</td>
</tr>
<tr>
<td>add_8/U1_7/CO (FA1A)</td>
<td>1.17 11.32 f</td>
</tr>
<tr>
<td>U2/Z (EO)</td>
<td>1.06 12.38 f</td>
</tr>
<tr>
<td>cout (out)</td>
<td>0.00 12.38 f</td>
</tr>
<tr>
<td>data arrival time</td>
<td>12.38 f</td>
</tr>
</tbody>
</table>

(Path is unconstrained)
- Shows how report_timing calculates delay of a timing arc (cell or net)
- Use to understand contributors to delay calcs
Additional Commands

all_connected
all_fanin
all_fanout
all_registers
get_attribute
report_area
report_attribute
report_cell
report_hierarchy
report_net
report_path_group
report_resources
report_timing_requirements
report_transitive_fanin
report_transitive_fanout

Analysis Summary

- **Before optimization**
  - Verify user-defined constraints and attributes
  - Check design consistency and timing path integrity

- **During optimization**
  - Optionally customize compile log
  - Optionally checkpoint compile run for debugging

- **After optimization**
  - Verify timing and DRC constraints are satisfied
Take a Break!