Logic Synthesis

CIC 楊智喬
Tel: (03)5773693 ext:154
Email: ccyang@cic.org.tw

Schedule (day 1)

- Introduction to Logic Synthesis
  - Introduction
  - Design object
  - Static Timing Analysis (STA)
  - Synopsys design analyzer environment
- HDL Coding For Synthesis
  - Synthesizable Verilog HDL
  - Some tricks in Verilog HDL
  - Designware library
- Lab Time (Lab1)

Schedule(day 2)

- Design Constraint
  - Setting design environment
  - Setting design constraint
- Design Optimization
  - Compile the design
  - Finite state machine optimization
- Synthesis Report and Analysis
- Lab Time (Lab2)

Introduction

Introduction to Logic Synthesis

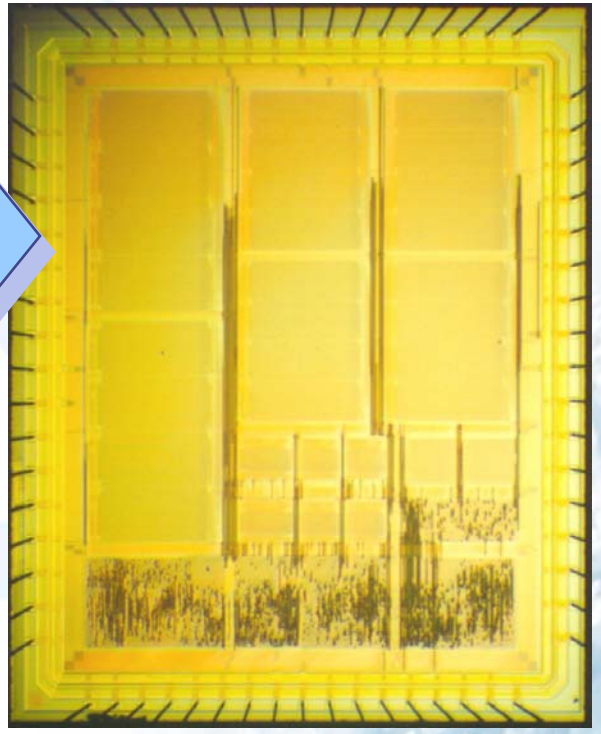
- Introduction
- Design object
- Static Timing Analysis (STA)
- Synopsys design analyzer environment

Introduction

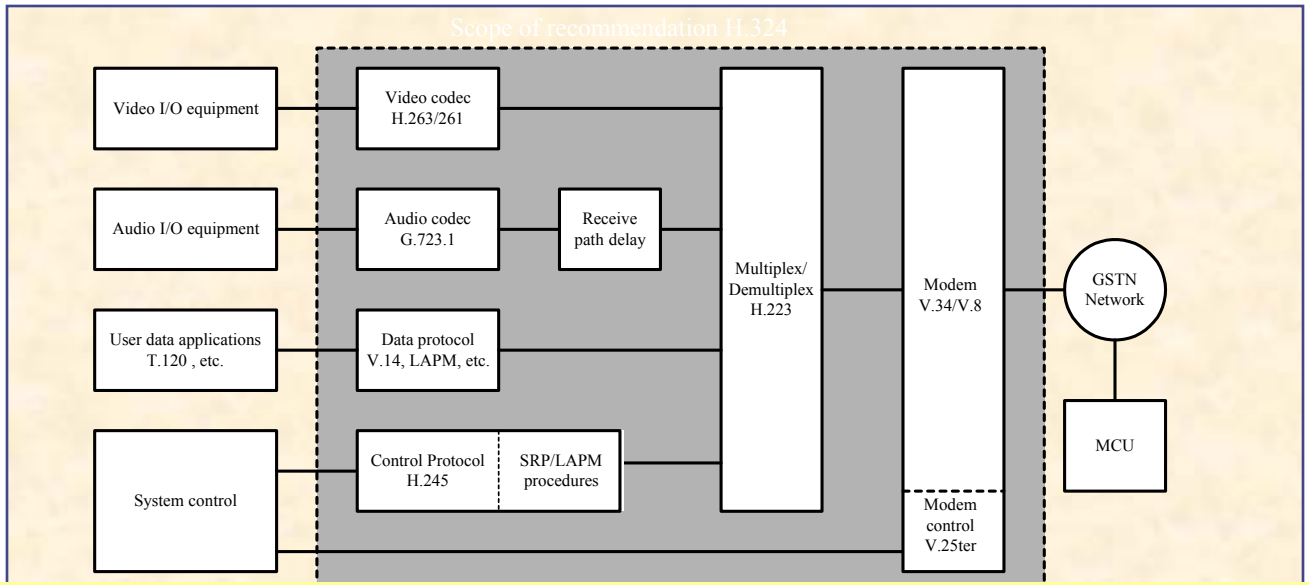
IC Design and Implementation

Idea

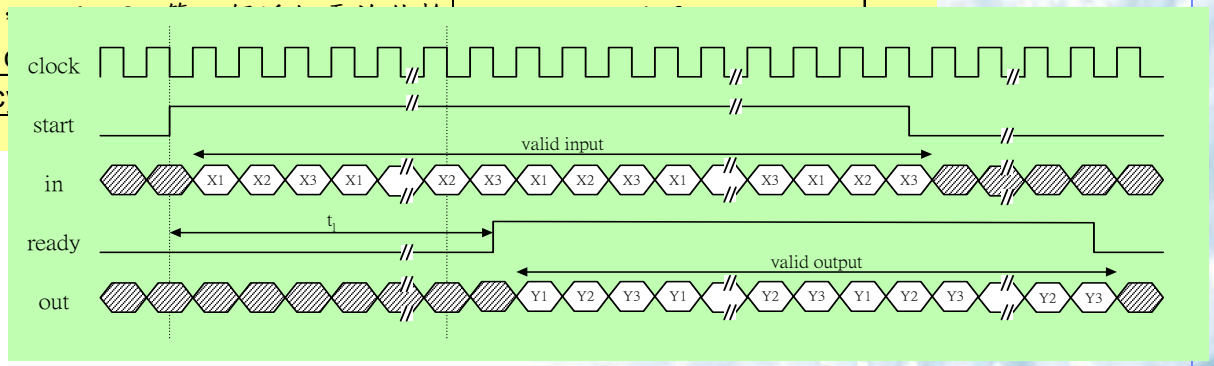
Chip



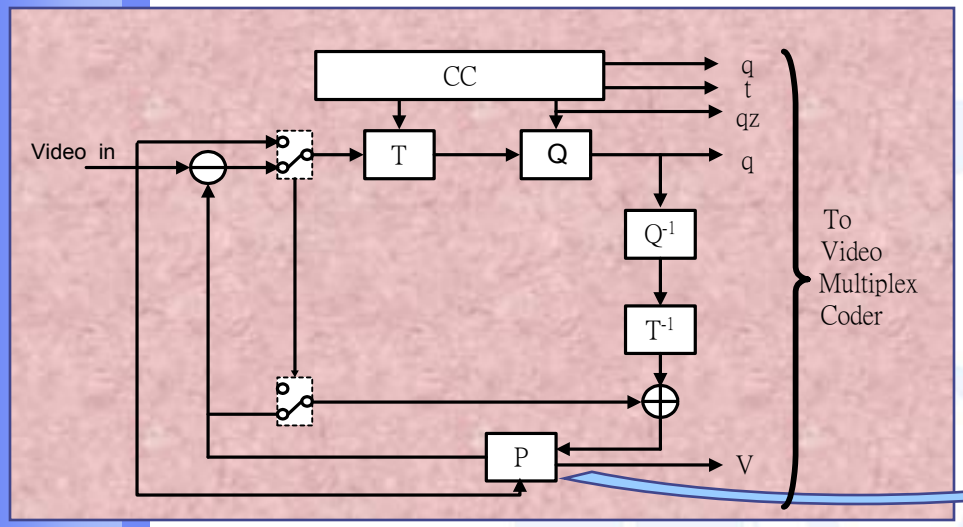
System Spec.



腳位名稱		描述	Drive Strength/Output Load
clk	輸入	系統時脈	assume infinite
reset	輸入	系統重置訊號，high active	1 ns/pf
din	輸入	每個clock cycle輸入一個16-bit 正整數	1 ns/pf
ready	輸出	reset為1時 出前的半個	
dout	輸出	每個clock c	



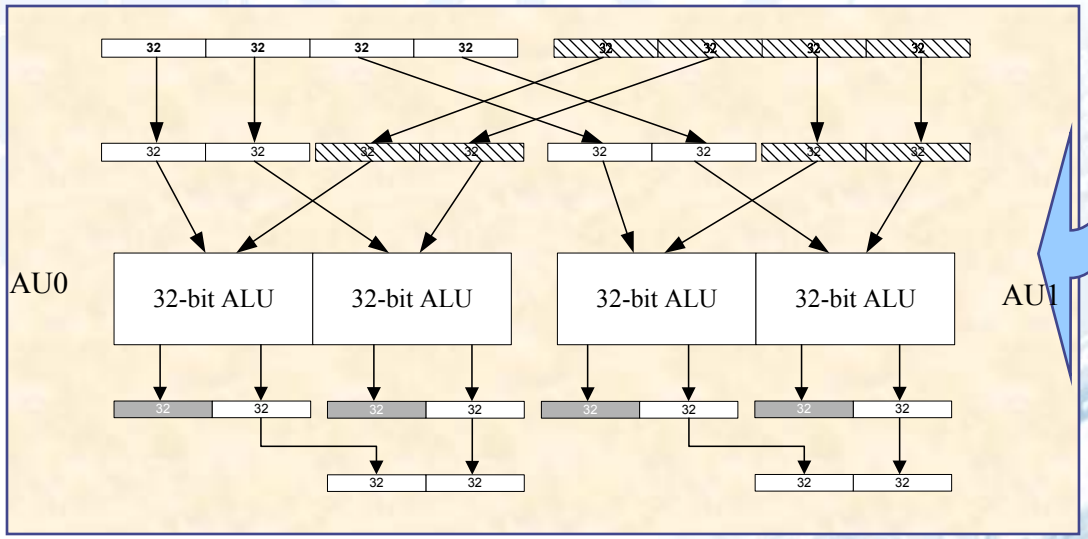
Algorithm Analysis



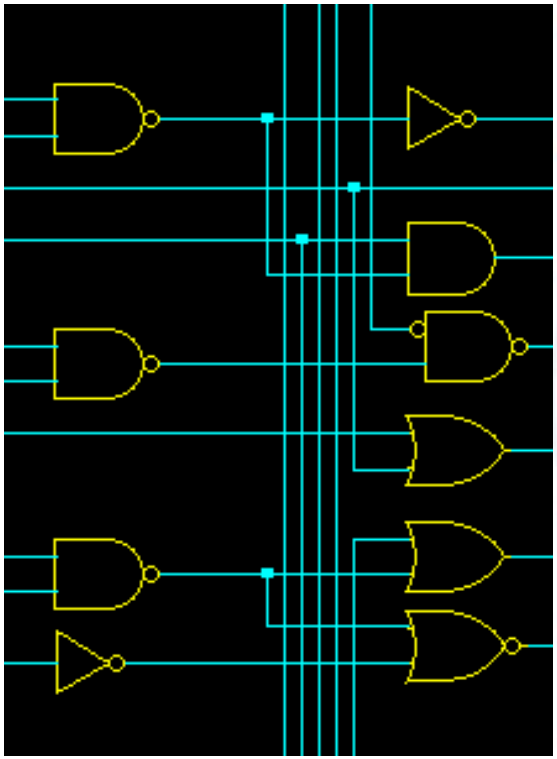
System Level

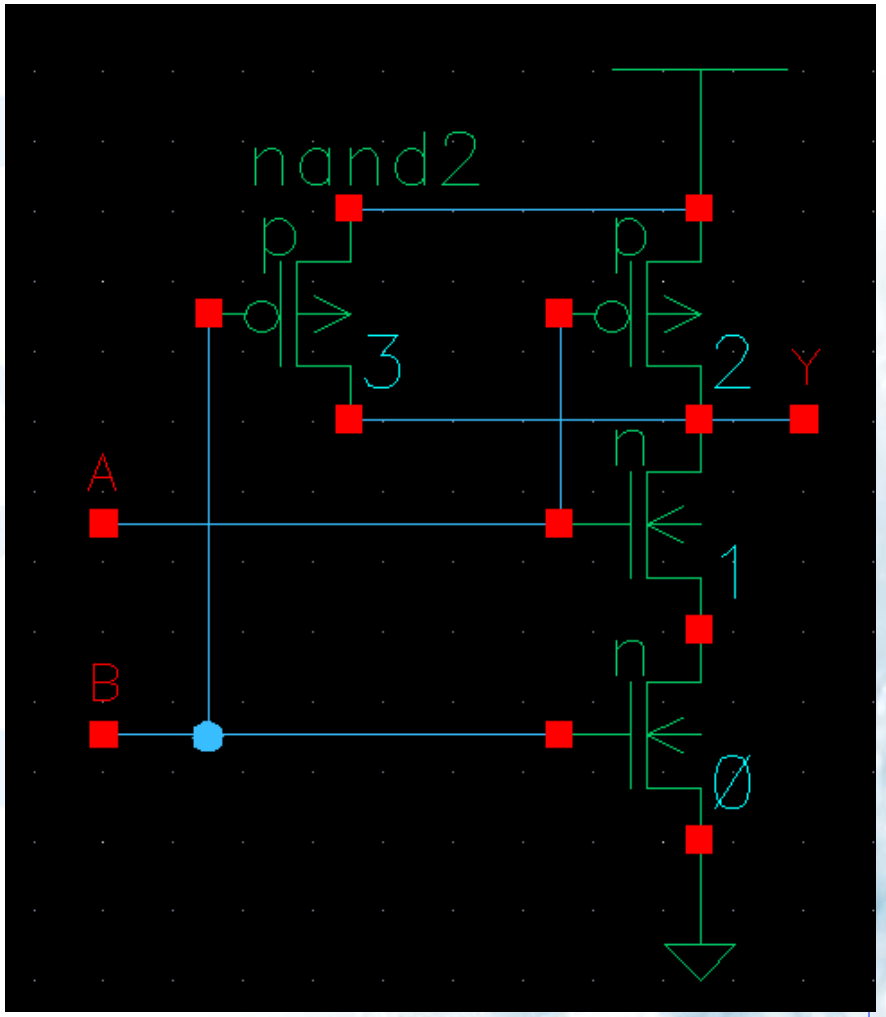
$$sad = \min SAD(k,l) = \min \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \left| I_c(i,j) - I_r(i+k, j+l) \right|$$

RTL Level

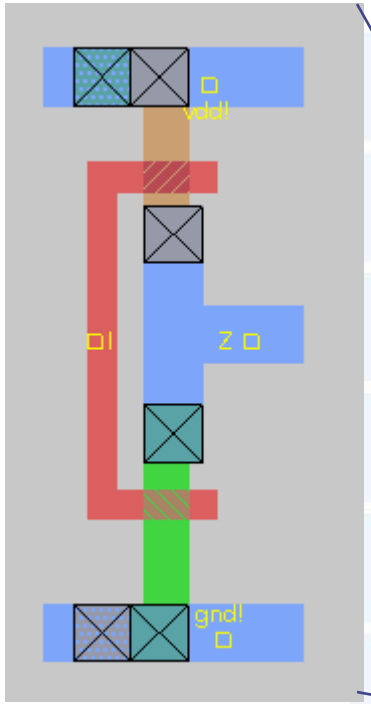


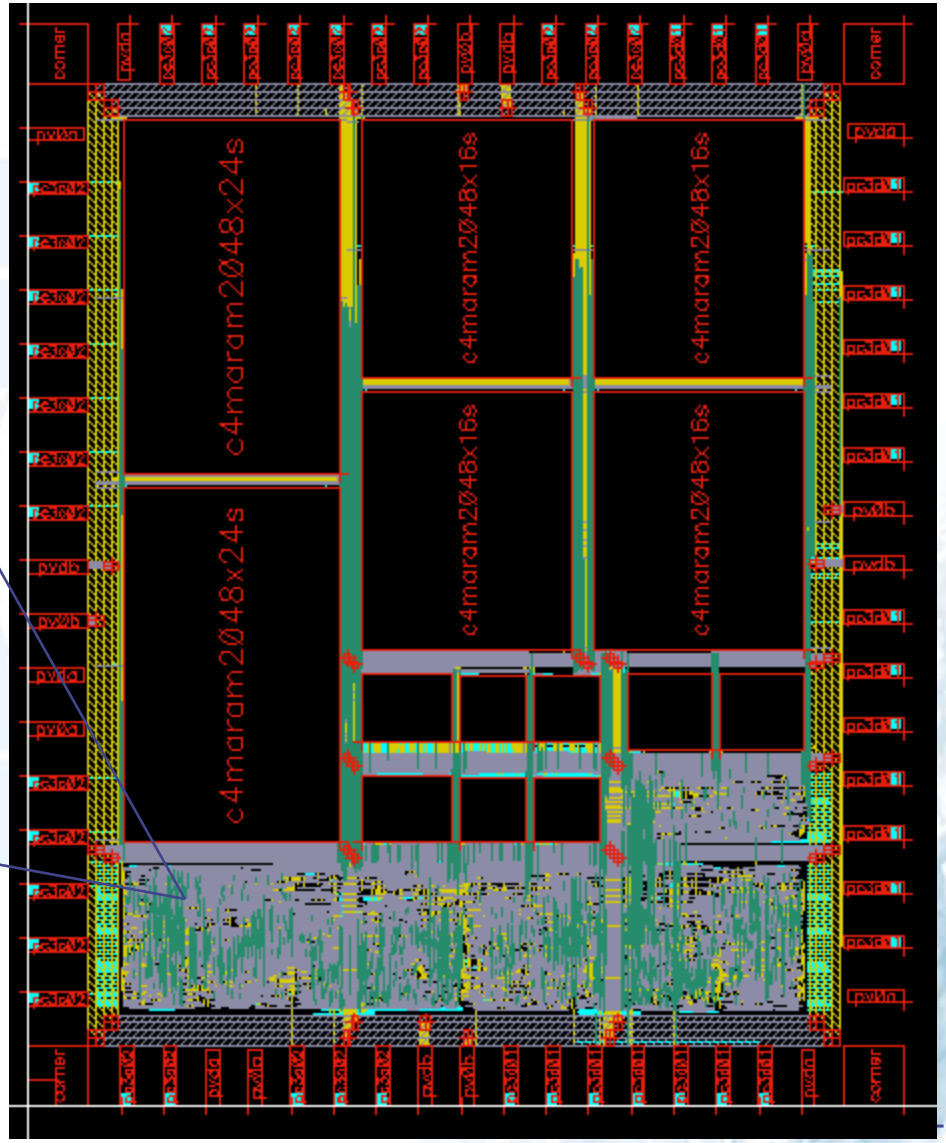
Gate and Circuit Level Design



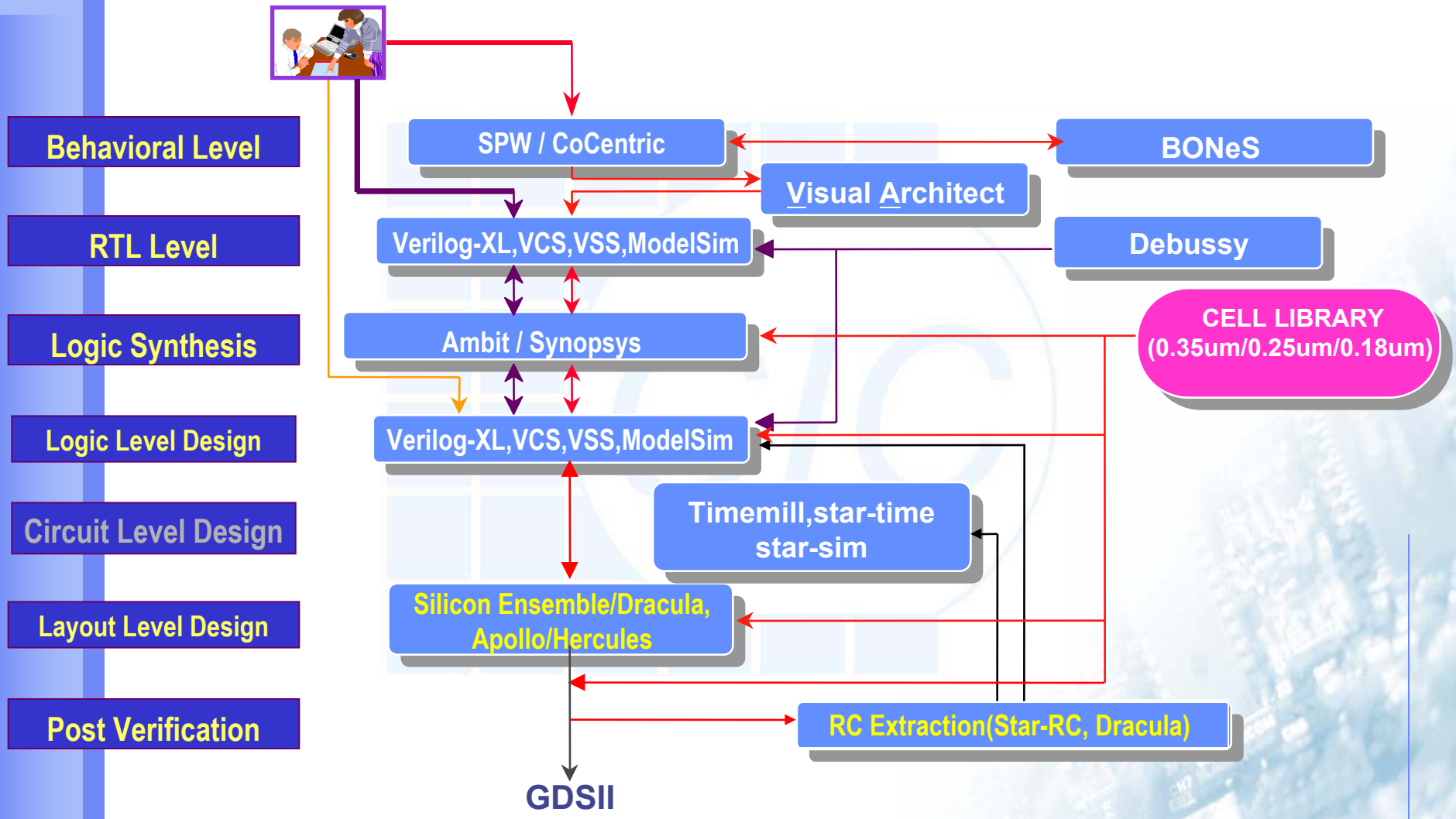


Physical Design





Cell-Based Design Flow



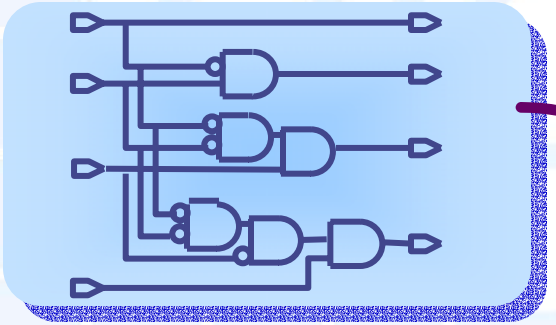
What is Synthesis

- Synthesis = translation + optimization + mapping

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

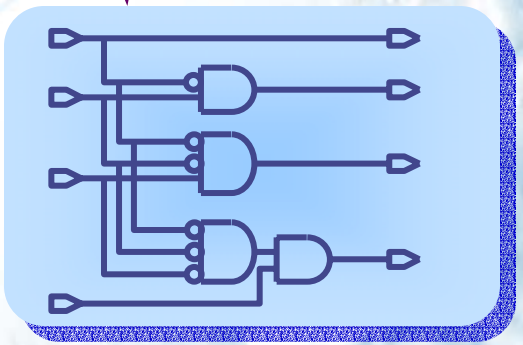
HDL Source

Translate



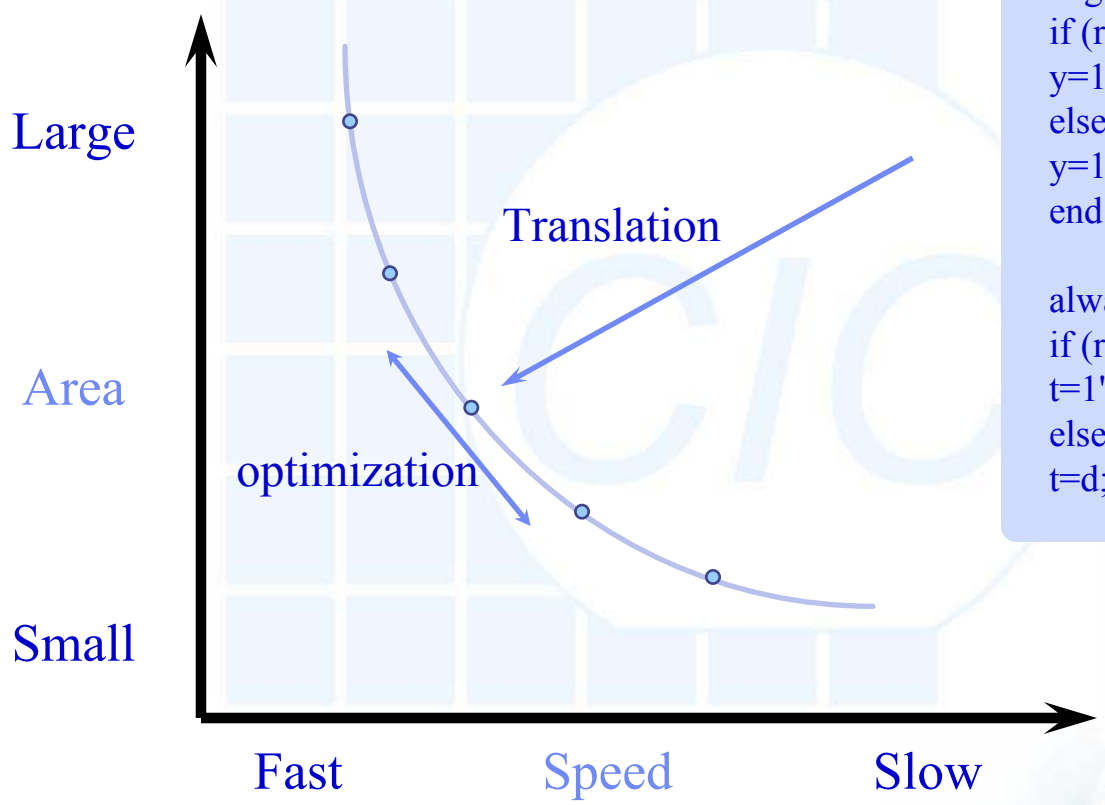
Generic Boolean (GTECH)

Optimize + Map



Target Technology

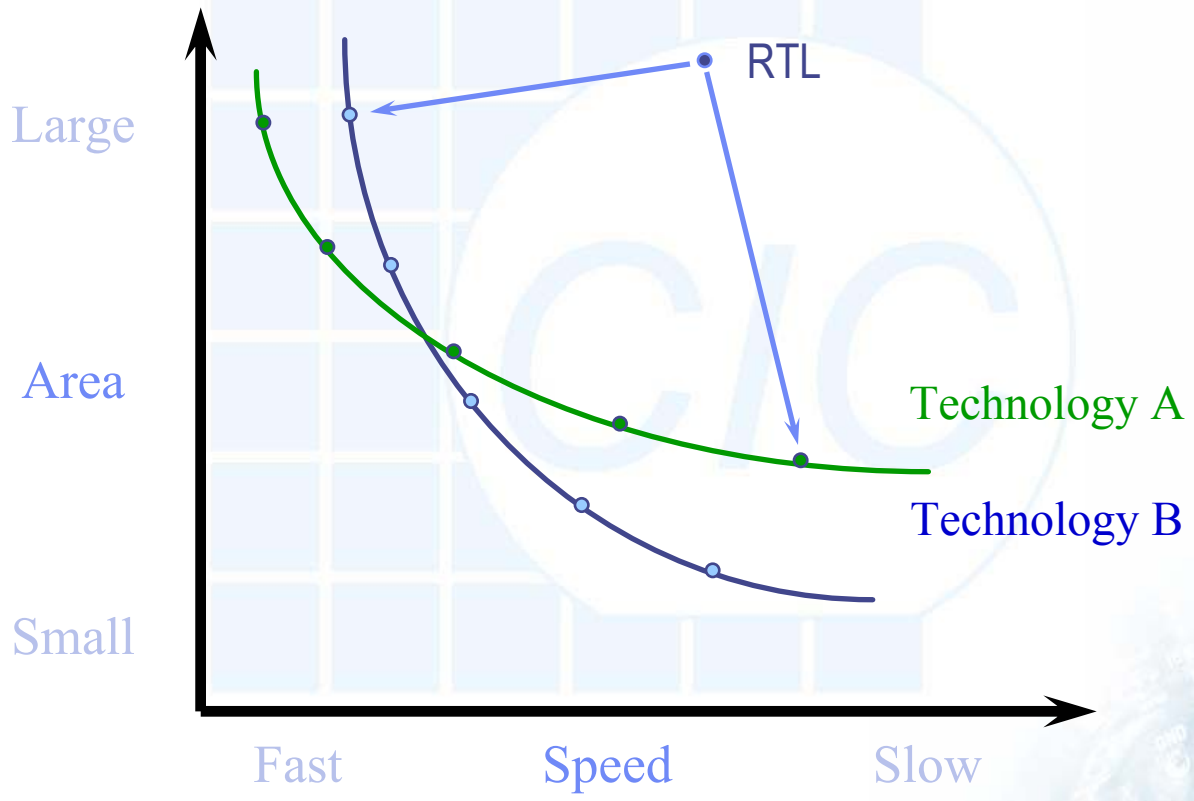
Synthesis is Constraint Driven




```
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;
```

Technology Independent

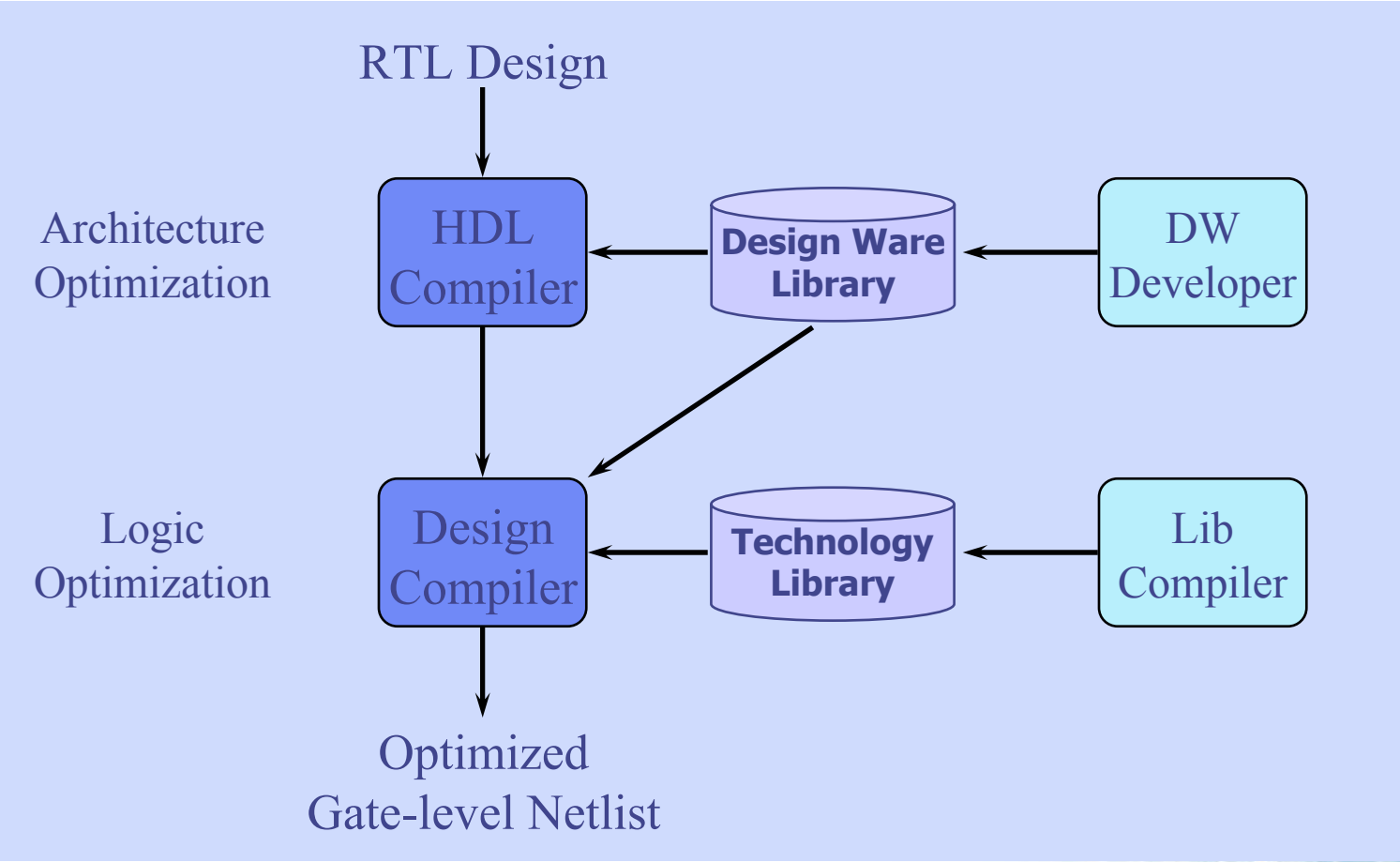
- Design can be transferred to any technology



Tools We will Use

Tool	Purpose
Design Analyzer	User graphical interface of synopsys synthesis tool
HDL Compiler	Translate Verilog descriptions into Design Compiler
Design Compiler	Constraint driven logic optimizer
Design Time	Static Timing Analysis (STA) engine
Design Ware (*)	Enable synthesis using DesignWare library

Logic Synthesis Overview

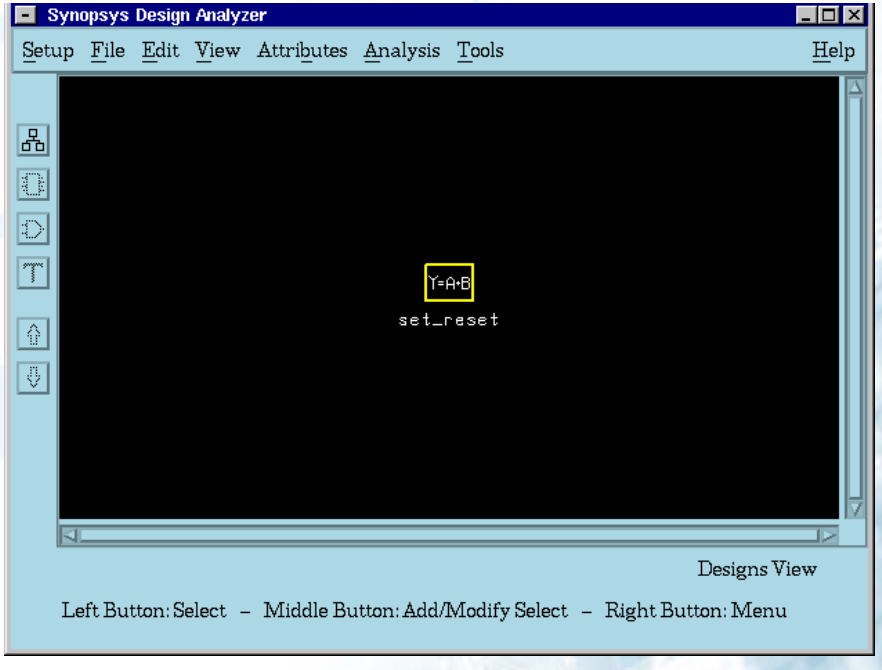


HDL Compiler (1/2)

HDL Compiler

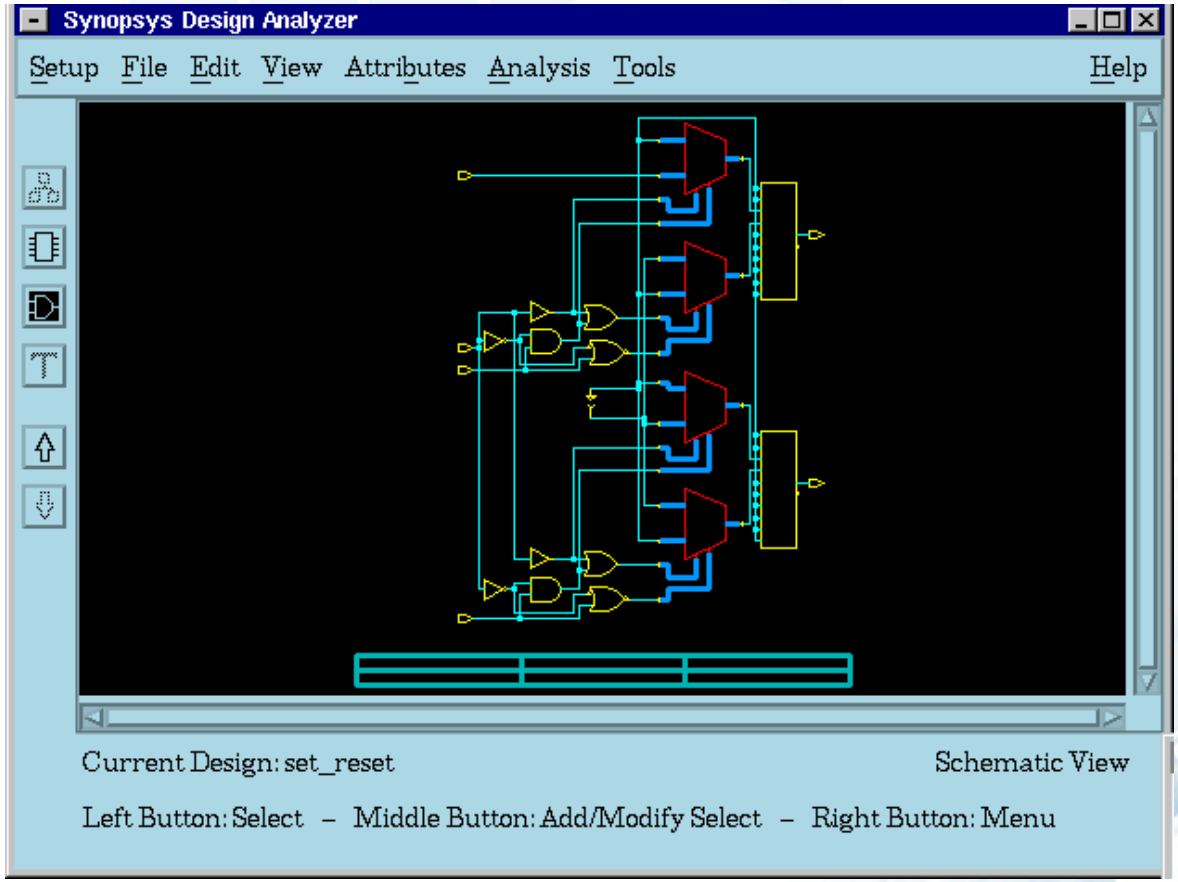
```
always @(reset or set)
begin : direct_set_reset
if (reset)
y=1'b0;
else if (set)
y=1'b1;
end

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

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



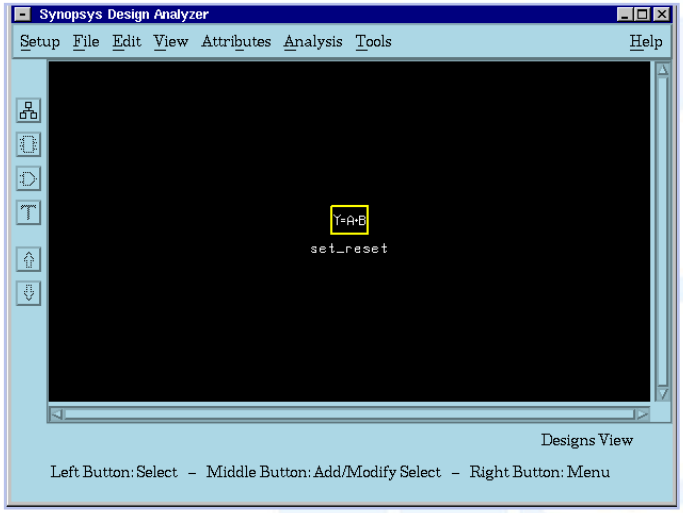
HDL Compiler (2/2)

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



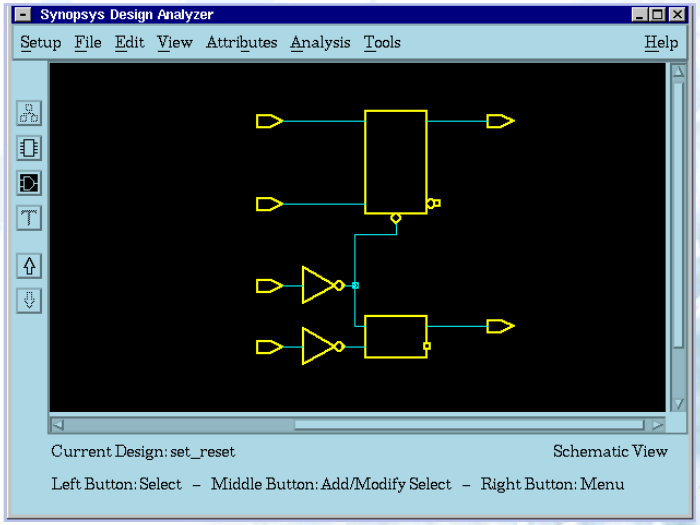
Design Compiler

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



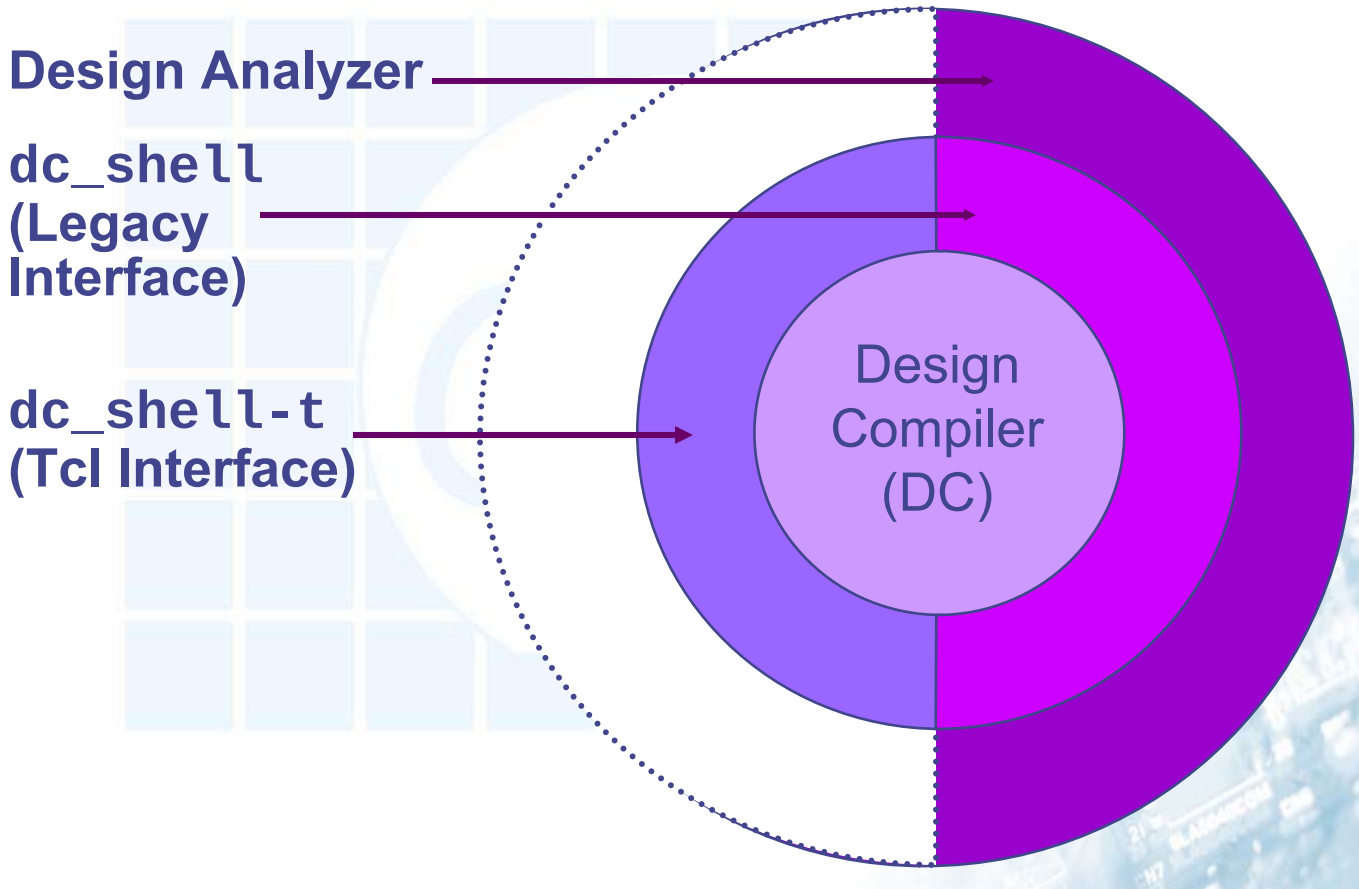
+

=



Design Compiler Interaction

- Three ways to interface



Synopsys Related Files

Files	Purpose
.cshrc	set path and environment variables and license check
.synopsys_dc.setup	Three distinct files are read and executed when DC is invoked ***1st***. system-wide (do not modify): (e.g. $SYNOPSYS/admin/setup/) ***2nd***. User's home directory (e.g. ~ccyang/) ***3rd.*** User's current working directory (e.g. ~ccyang/dc/)

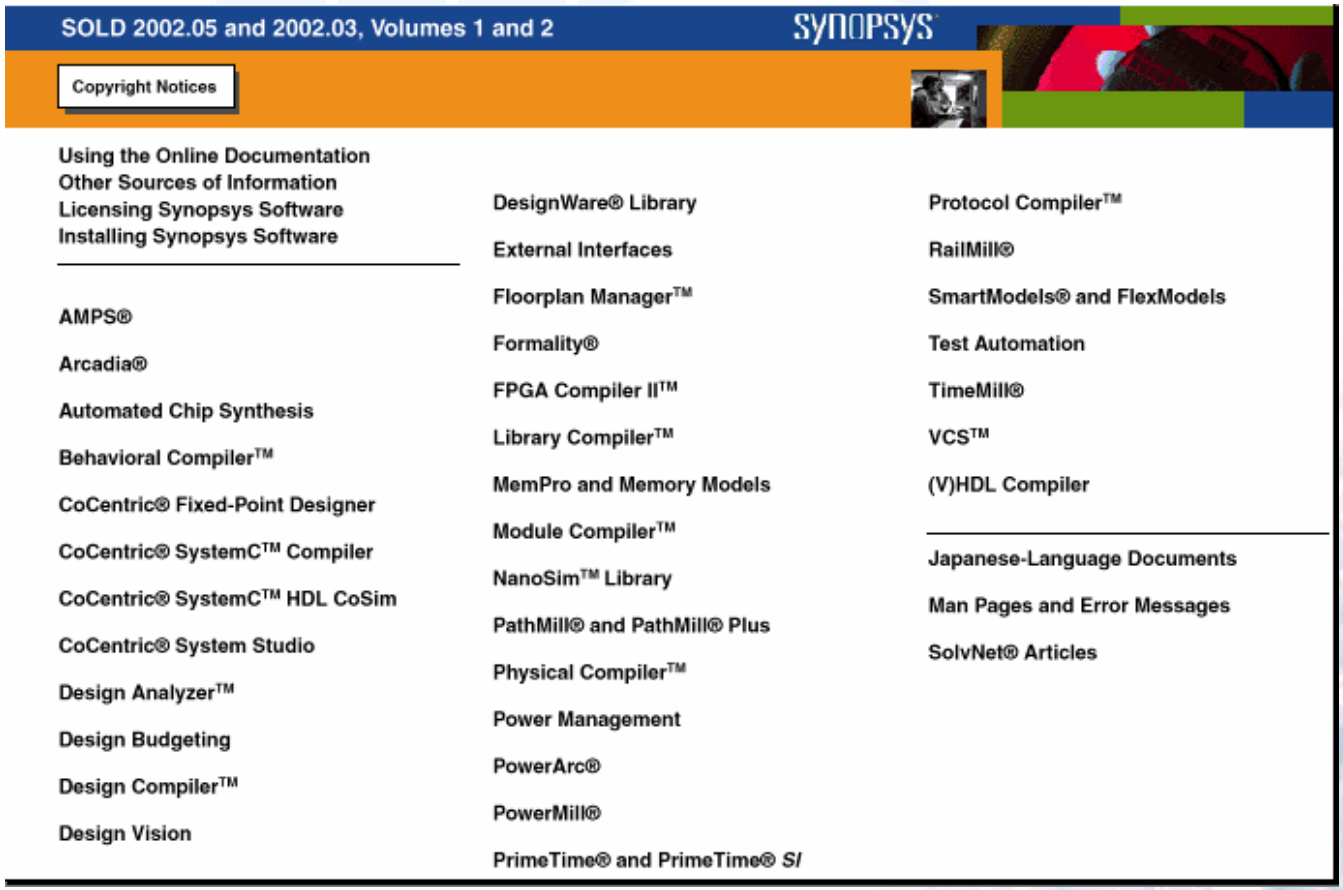
- ***NOTE***
  - These 3 files are always read ***in the same order.***
  - Any repeated command can ***override*** the previous one.

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

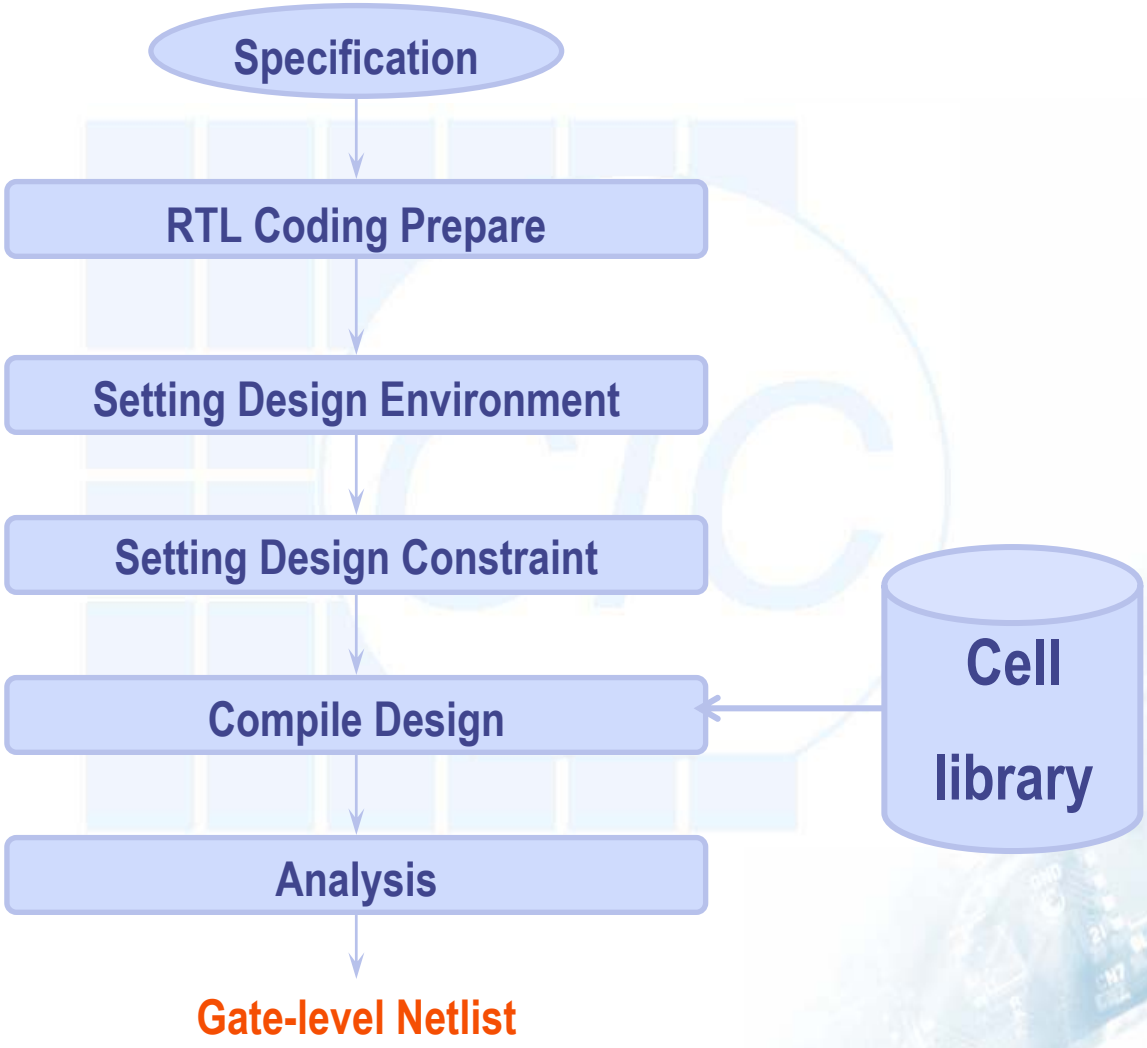
- link_library: the library used for interpreting input description
  - Any cells instantiated in your HDL code
  - Wire Load or Operating Condition models used during synthesis
- target_library: the ASIC technology that the design is mapped to
- symbol_library: used during schematic generation
- search_path: the path to search for unsolved reference library or design
- synthetic_library: designware library to be used
- Other variables

.synopsys_dc.setup file

- In CIC cell_based flow, we support compass 0.35um cell library, the .synopsys_dc.setup file is as follows

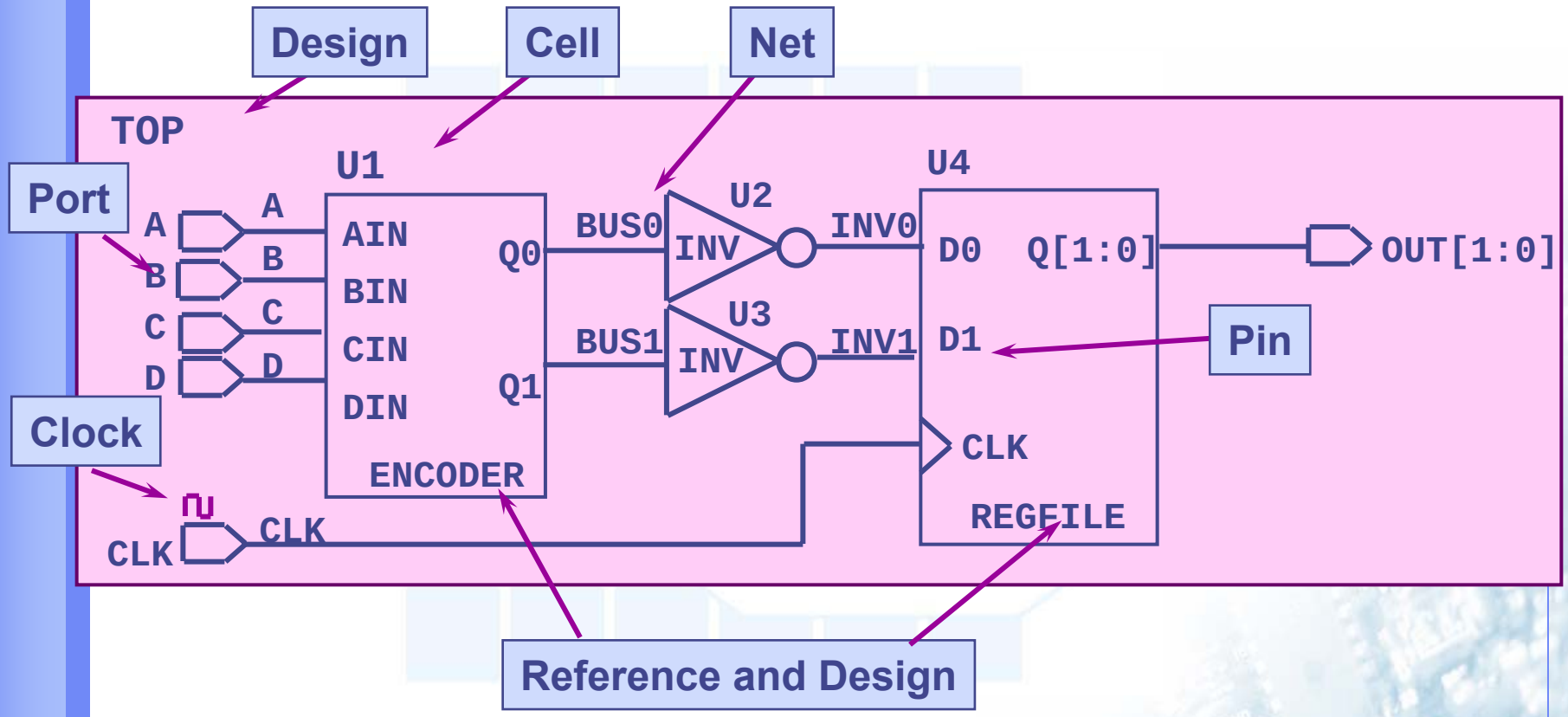
```
search_path = {. /Your_path/CIC_CBDK35_V3/Synopsys}+search_path;
link_library = {“*”, “cb35os142.db”, “dw_foundation.sldb”} ;
target_library = {cb35os142.db};
symbol_library = {generic.sdb} ;
synthetic_library =  {“dw_foundation.sldb”};

hdlin_translate_off_skip_text = "TRUE"
edifout_netlist_only = "TRUE"
verilogout_no_tri = true ;
plot_command = "lpr -Plp" ;\
view_script_submenu_items = \
{"Avoid assign statement", "set_fix_multiple_port_nets -all -buffer_constant",\
 "Change Naming Rule", "change_names -rule verilog -hierarchy", \
"Write SDF", "write_sdf -version 1.0 -context verilog chip.sdf"}
```

ASIC Synthesis Design Flow



- Synthesis Object

Design Objects (Schematic Perspective)



Design Objects

Seven Types of Design Objects:

Design:	A circuit that performs one or more logical functions
Cell:	An *instance* of a design or library primitive within a design
Reference:	The name of the original design that a cell instance "points to"
Port:	The input or output of a *design*
Pin:	The input or output of a *cell*
Net:	The wire that connects ports to pins and/or pins to each other
Clock:	A timing reference object in DC memory which describes a waveform for timing analysis

Design Objects (Verilog Perspective)

Design

Clock

Port

Net

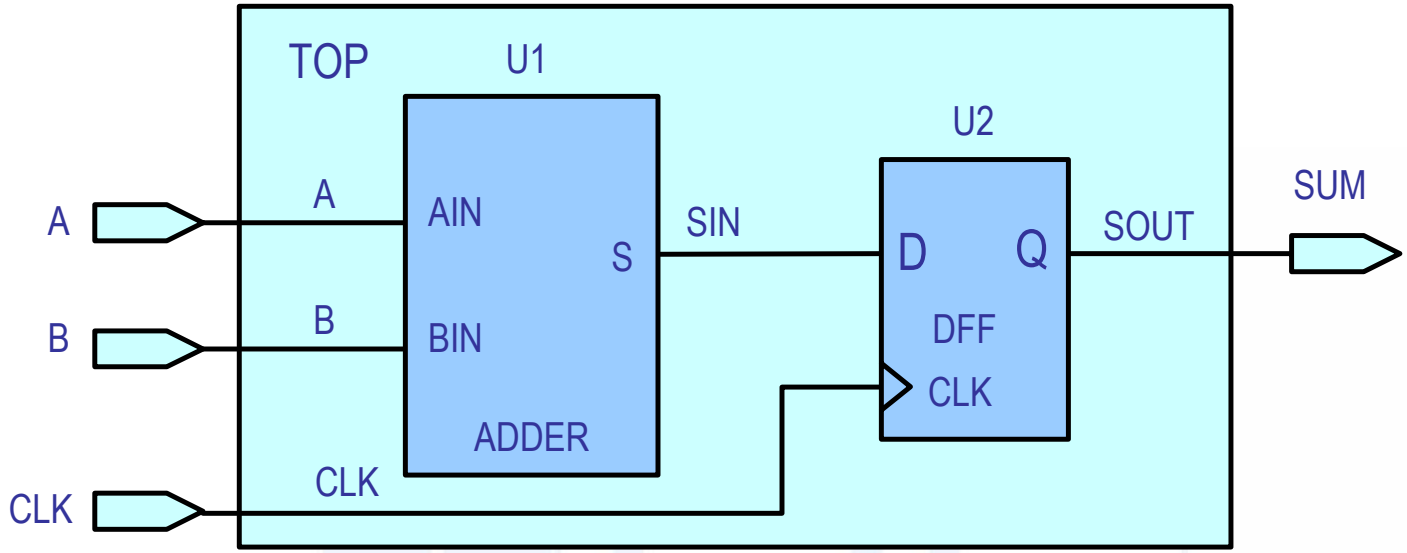
Reference

Cell

Pin

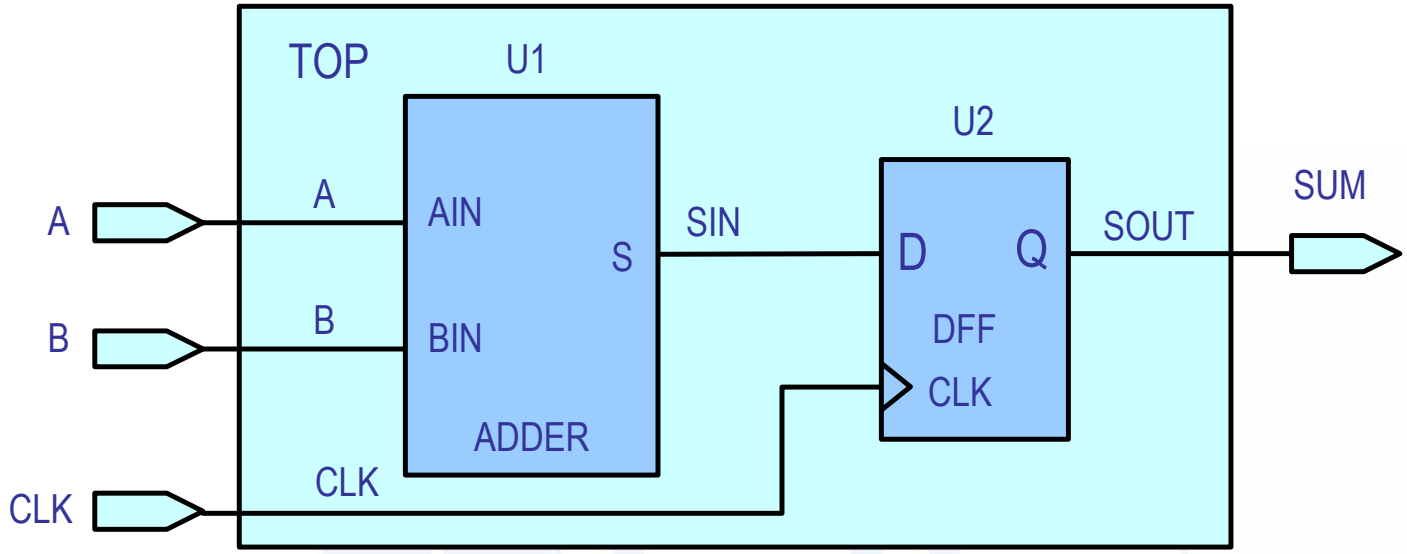
```
module TOP (A,B,C,D,CLK,OUT1);
  input A, B, C, D, CLK;
  output [1:0] OUT1;

  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
```

Design Objects Exercise



- Make a list of all the ports in the design?{ }
- Make a list of all the cells that have the letter "U" in their name? { }
- Make a list of all the nets ending with "CLK"? { }
- Make a list of all the "Q" pins in the design? { }
- Make a list of all the references? { }

Design Objects Exercise



- Make a list of all the ports in the design?{ A, B, CLK, SUM }
- Make a list of all the cells that have the letter “U” in their name? { ADDER/U1, DFF/U2 }
- Make a list of all the nets ending with “CLK”? { CLK }
- Make a list of all the “Q” pins in the design? { U2/Q }
- Make a list of all the references? { ADDER,DFF }

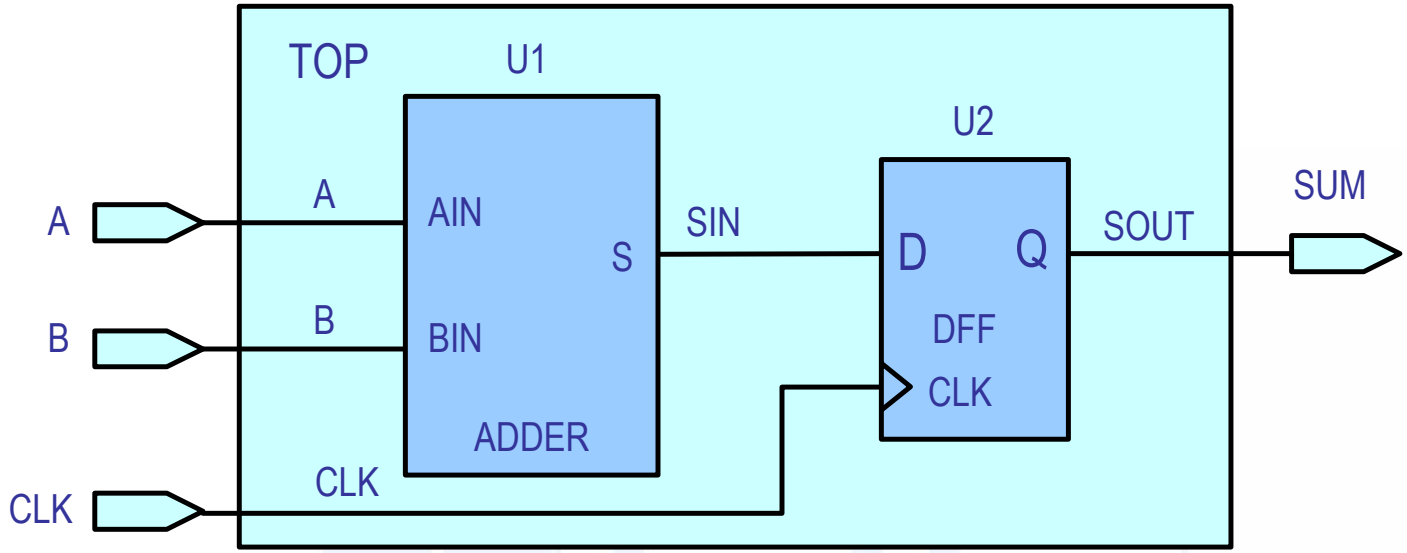
`find` Command Syntax

- Search the ***current design*** for names of the given object type.
  - Can be used stand-alone or composed with other functions.
- Syntax: `find type [name_list] [-hierarchy]`
- `type`: *design, port, reference, cell, clock, pin or net*
- `name_list` (optional):
  - List of design of library object names, Use brackets ({list}) for multiple names.
  - If no name_list is given, the find command lists all the names of specified object type.
- `-hierarchy` (optional):
  - Use this option if all objects within a hierarchical design are to be returned. Only works with these object types: design, net, cell or pin.

find Command Syntax

- List all the ports of the current design:
  - `find (port, "*")`
- List all the instances that start with the letter "B" or "D"
  - `find (cell, {B* D*})`
- Find the nets "n2003", "n2004"
  - `find (net, {"n2003", "n2004"})`
- Place a dont_touch attribute on all the designs in the hierarchy
  - `set_dont_touch find (design, "*" -hier)`
- List all the pins of the "FD1" cell of the "class" library
  - `find (pin class/FD1/*)`

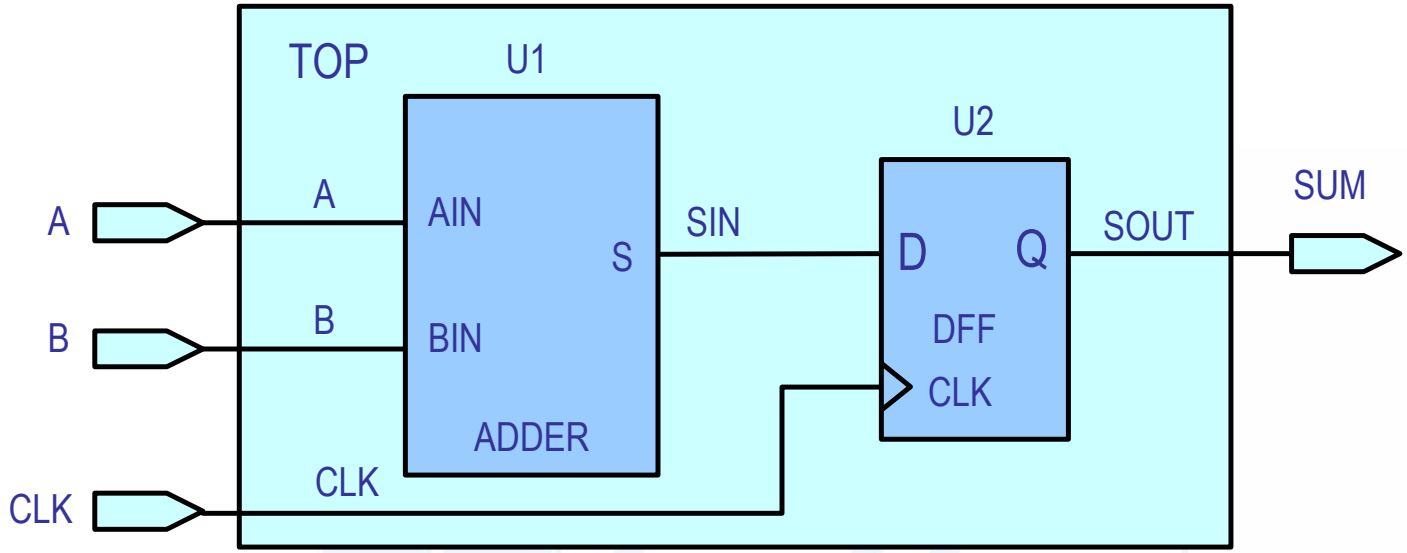
find Command Exercise



Write find commands to do the following:

1. find a list of all the ports in the design?{ }
2. find a list of all the cells that have the letter "U" in their name? { }
3. find a list of all the nets ending with "CLK"? { }
4. find a list of all the "Q" pins in the design? { }

find Command Exercise



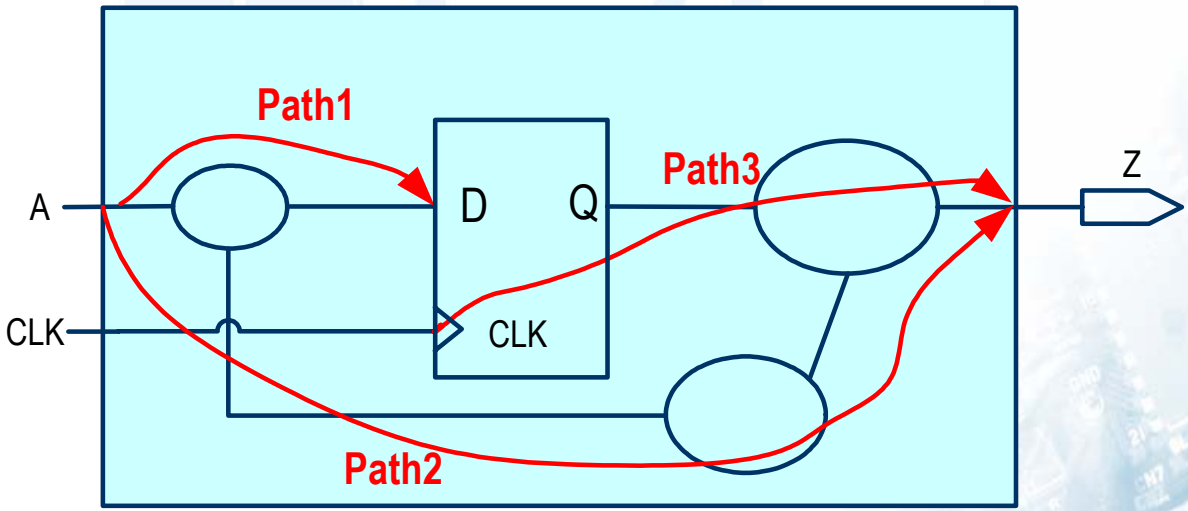
Write find commands to do the following:

- find a list of all the ports in the design? { find (port, "*") }
- find a list of all the cells that have the letter "U" in their name? { find (cell, "*U*") }
- find a list of all the nets ending with "CLK"? { find (net, "*CLK") }
- find a list of all the "Q" pins in the design? { find (pin, "*/Q") }

Static Timing Analysis

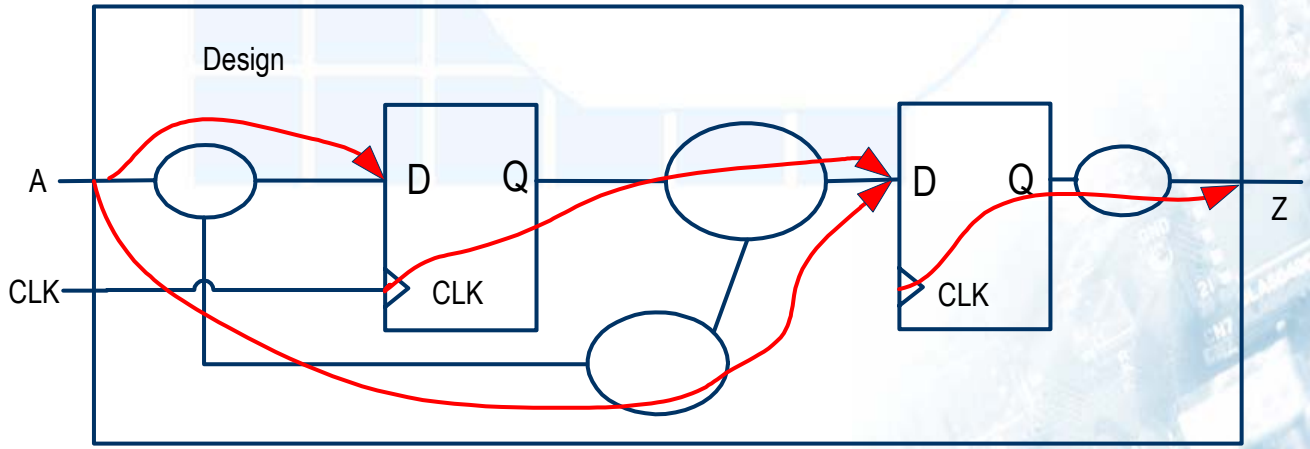
Static Timing Analysis (Design Time)

- A method for determining if a circuit meets timing constraints without having to simulate clock cycles.
  - Designs are broken down into sets of timing paths
  - The delay of each path is calculated
  - All path delays are checked to see if timing constraints have been met



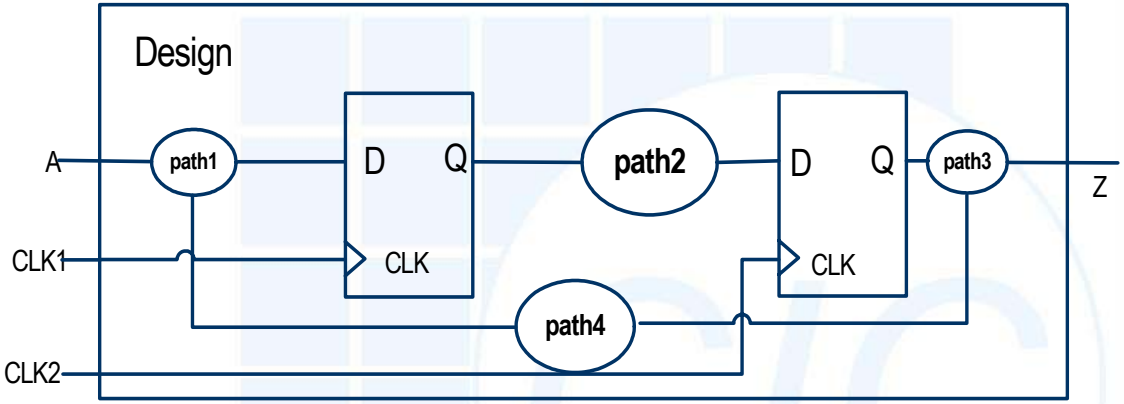
Timing Paths in Design Compiler

- Design Time breaks designs into sets of signal paths, each path has a startpoint and an endpoint.
  - Startpoints:
    - Input ports
    - Clock pins of sequential devices
  - Endpoints:
    - Output ports
    - Data input pins of sequential devices



Timing Groups

- How to organize timing paths into group?

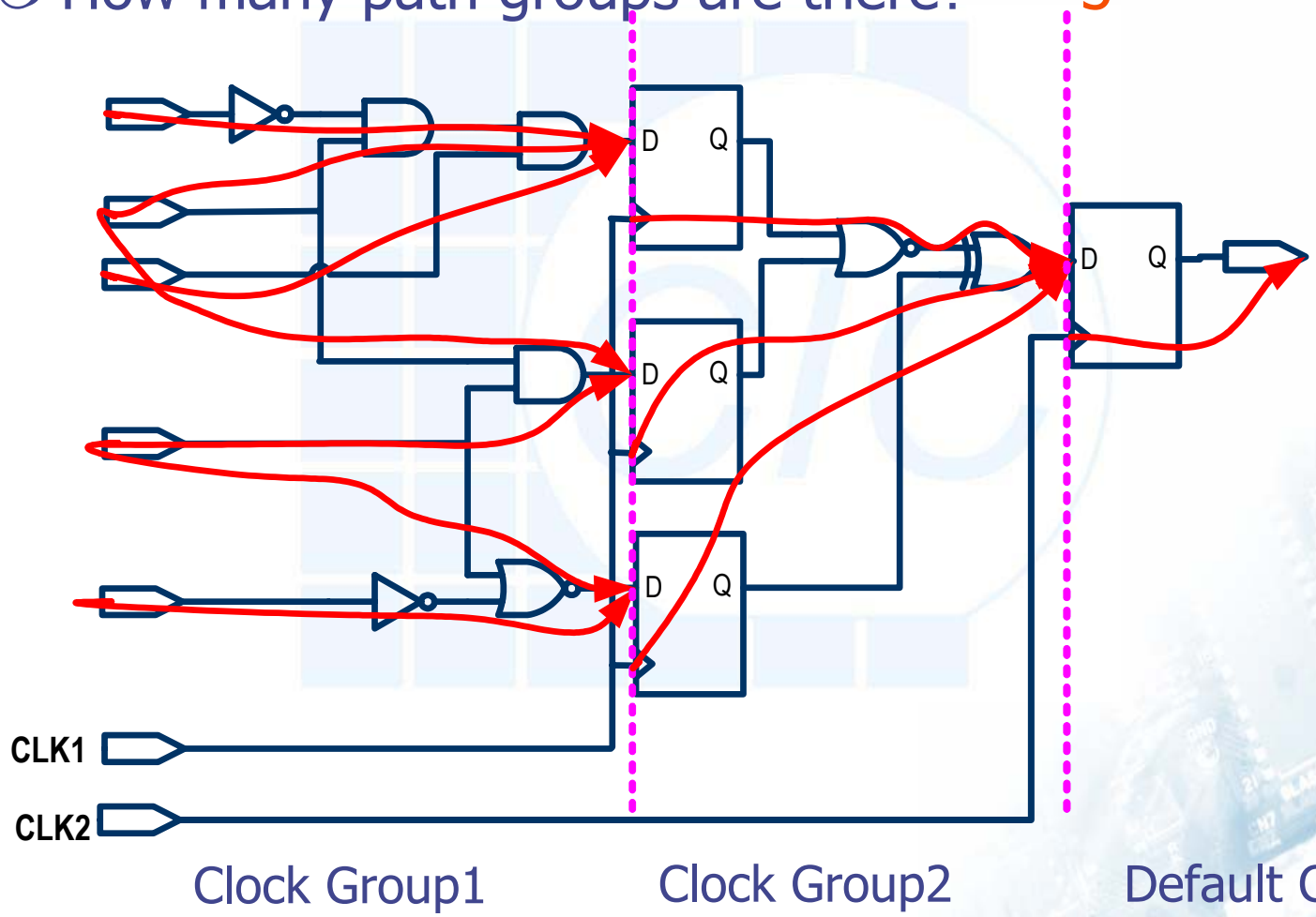


- Paths are grouped according to the clocks controlling their endpoints
- Each clock will be associated with a set of paths called a path group
- The default path group comprises all paths not associated with a clock

CLK1 path1 CLK2 path2 DEFAULT Path3 path4

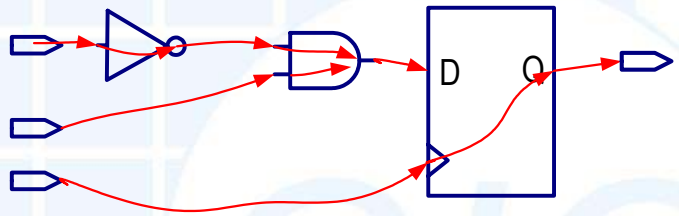
Timing Path Exercise

- How many timing paths do you see? 11
- How many path groups are there? 3

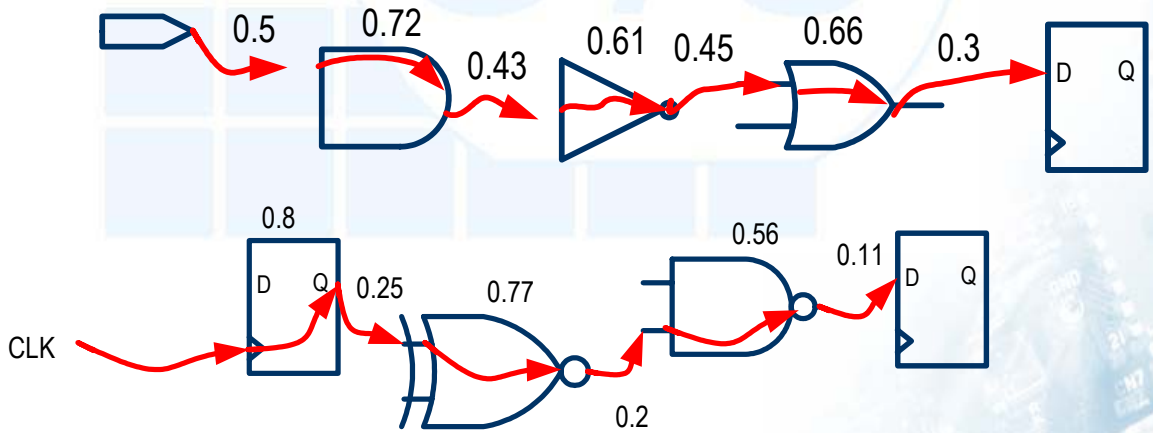


Schematic Converted To a Timing Graph

- To calculate total delay, Design Time breaks each path into timing arcs.
- Each timing arc contributes either a net delay or cell delay.



- Example of calculating a path delay
  - All the net and cell timing arcs along the path are added together



Path Delay = 0.8 + 0.25 + 0.77 + 0.2 + 0.56 +0.11 = 2.51 ns

Edge Sensitivity in Path Delay

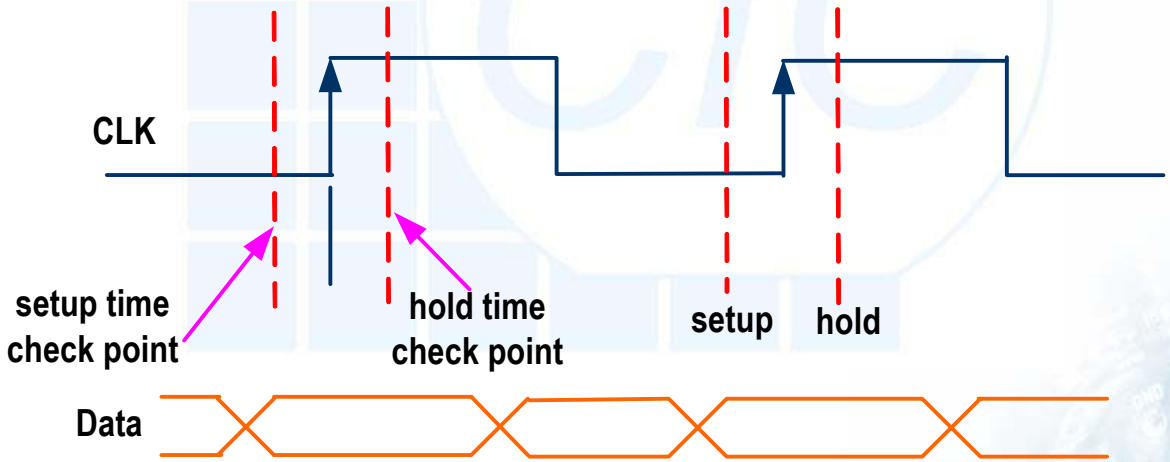
- There is an "Edge Sensitivity" (called unateness) in a cell's timing arc:
- Design Time keeps track of unateness in each timing path

```
library: pin (Z) {
     intrinsic rise: 1.5;
     intrinsic fall: 0.3;
}
```

```
library: pin (Z) {
     intrinsic rise: 1.5;
     intrinsic fall: 0.3;
}
```

Setup & Hold Time Check

- Setup Time: The length of time that data must stabilize before the clock transition
  - The maximum data path is used to determine if setup constraint is met
- Hold Time: The length of time that data must remain stable at the input pin after the active clock transition.
  - The minimum data path is used to determine if hold time is met

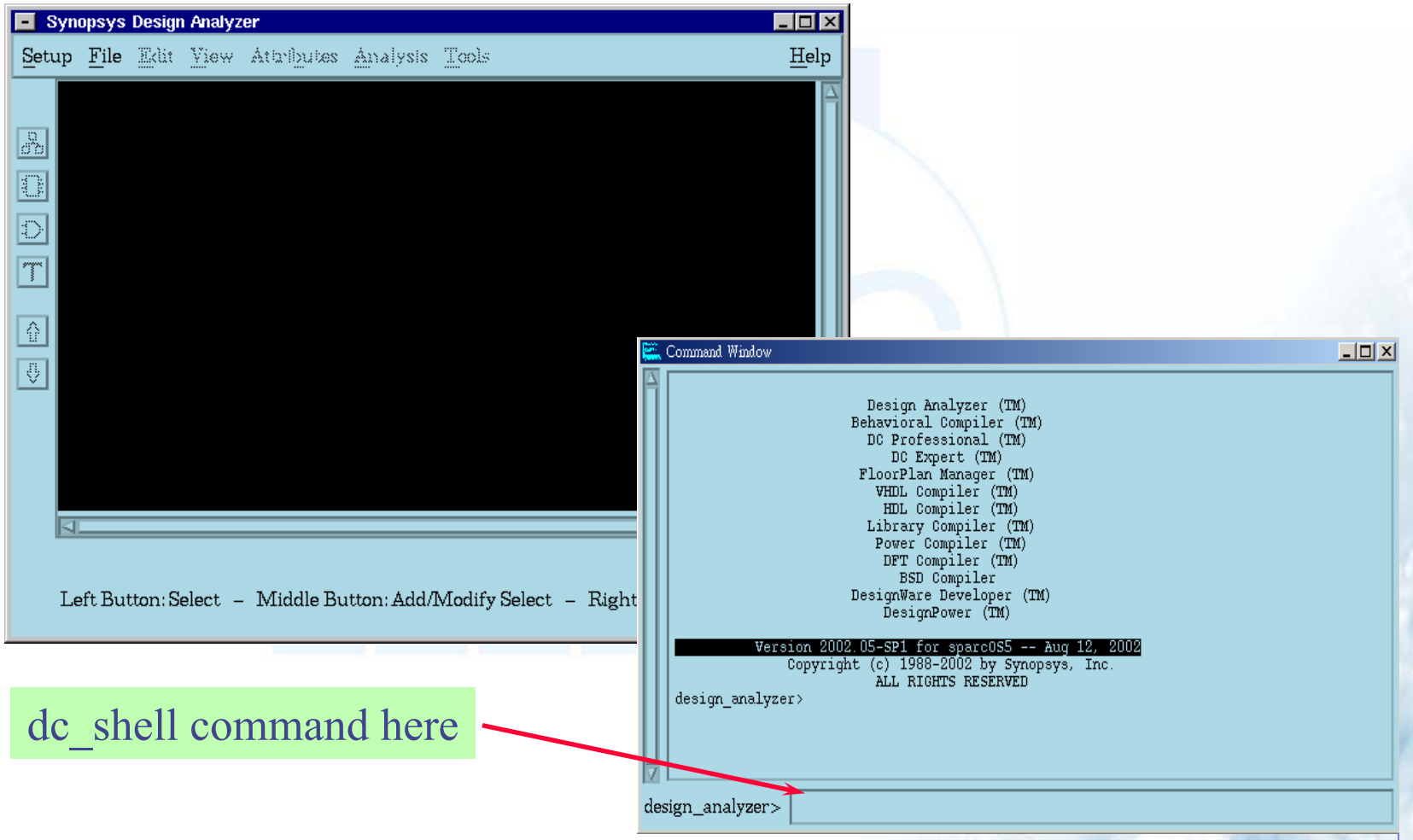


Synopsys Graphical Environment

Design Analyzer

Invoke Design Analyzer

- `Unix%> design_analyzer &`



dc_shell command here

Optimization Using the Design Analyzer

- 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)

What is a Design Library

- A Design Library is a *logical name* that maps to a UNIX directory used to store intermediate files produced by analyze, so that it won't clutter your present working directory
  - `Unix%> mkdir ~traina/analyzed"`
- Tell Design Compiler the logical library name and the UNIX directory, to associate with that logical library name
  - `define_design_lib logical_library_name -path unix_directory_path`

    (e.g. `define_design_lib WORK -path ~traina/analyzed`)
- To find out what is in a library
  - `report_design_lib logical_library_name`

    `report_design_lib WORK`

Analyze

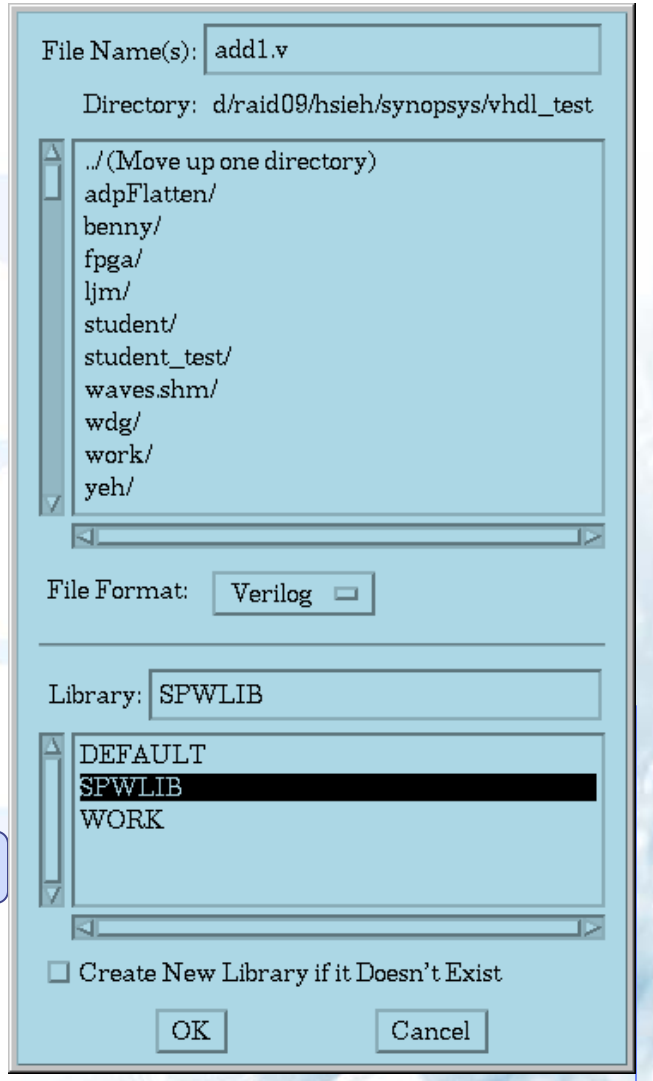
- Check VHDL & Verilog for syntax and synthesizability
- Create intermediate .syn and .mra files and places them in library specified -- design library

```
add1%verilog.syn
add1%verilog__verilog.syn
ADD1.mra
```

```
analyze -format verilog -library SPWLIB add1.v
```

- Equivalent to dc_shell command

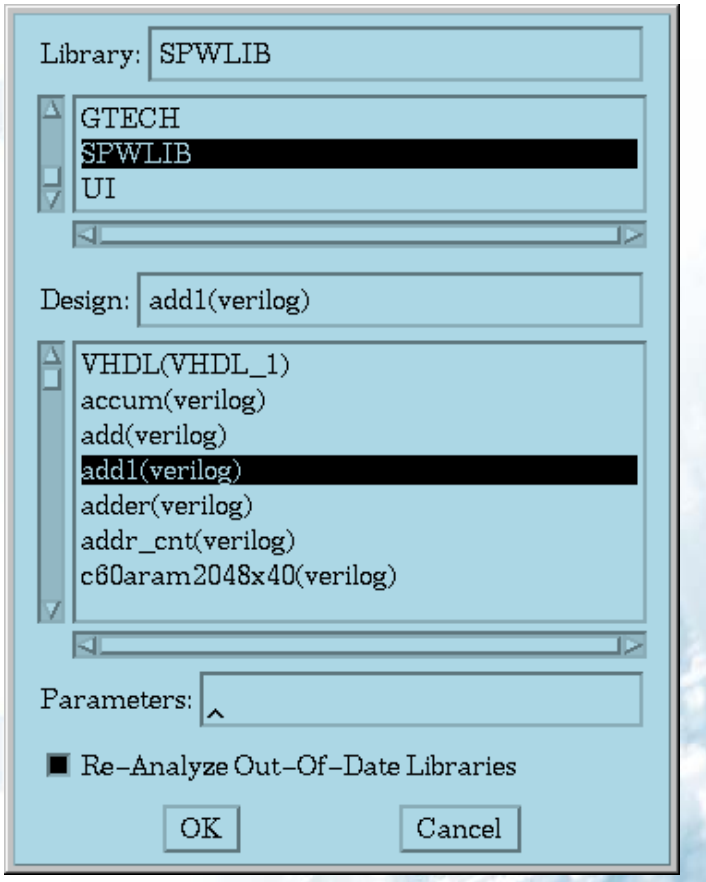
File/Analyze



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

File/Elaborate

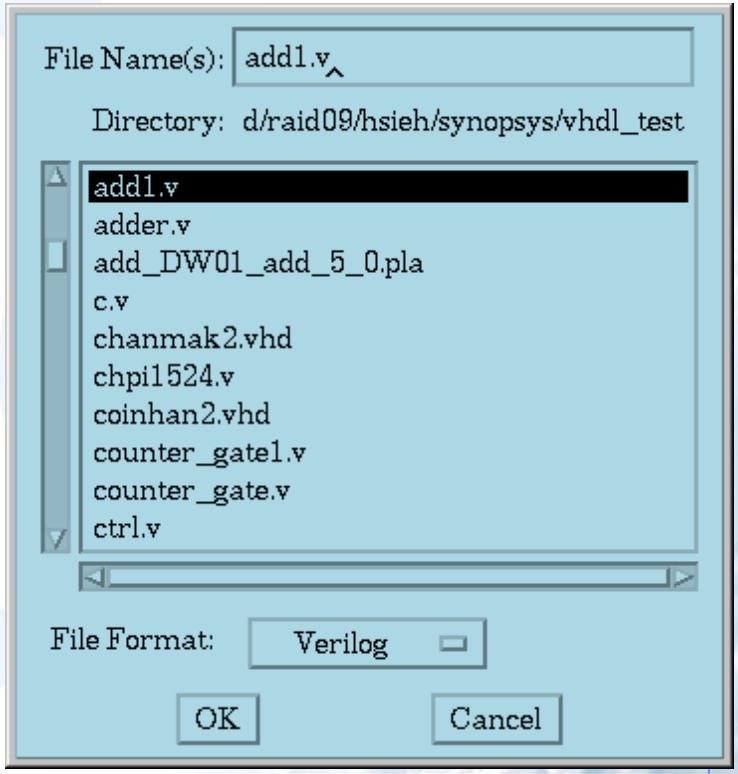


- Equivalent dc_shell command

```
elaborate add1 -architecture verilog -library SPWLIB -update
```

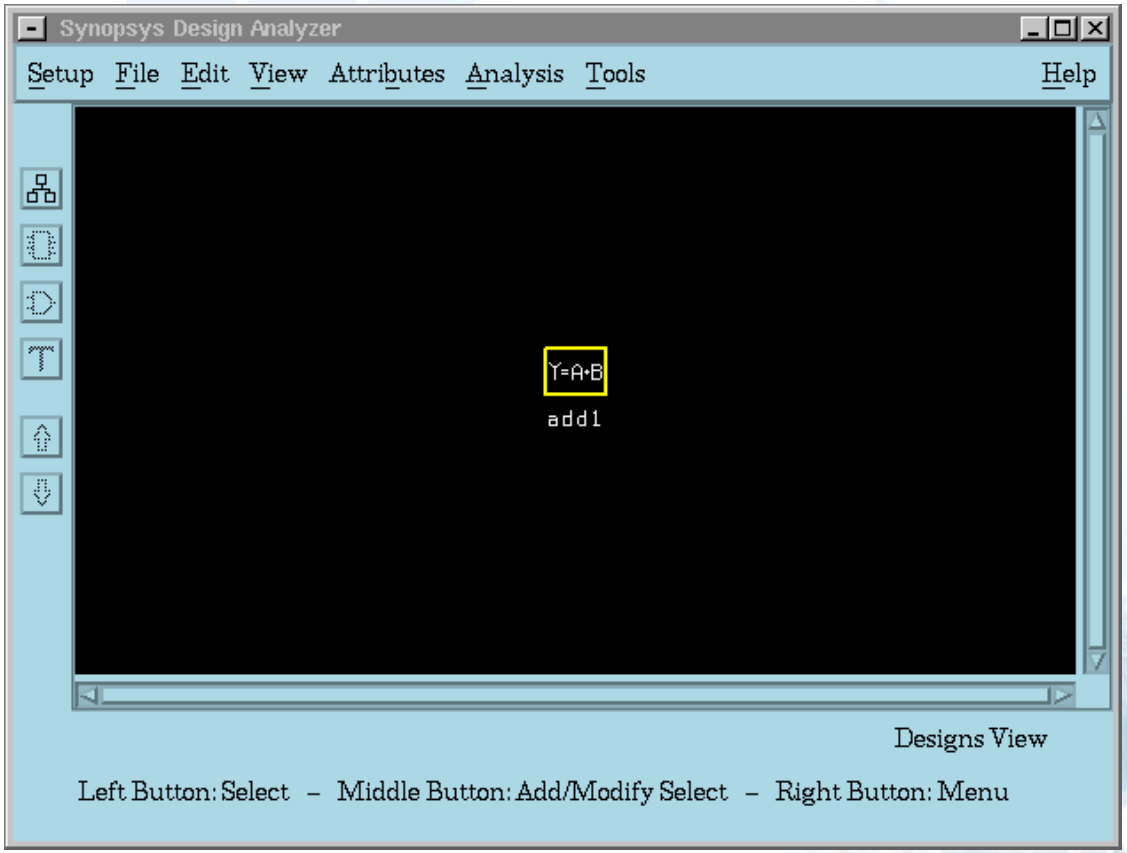
Read File

- 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



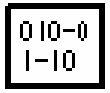
After Analyze/Elaborate or Read

- File/Analyze → File/Elaborate
- File/Read



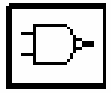
Four Types of Icon

EQUATION



PLA

FSM



NETLIST

EQUATION

Equation, or non-netlist VHDL or Verilog format

PLA

Programmable logic array format

FSM

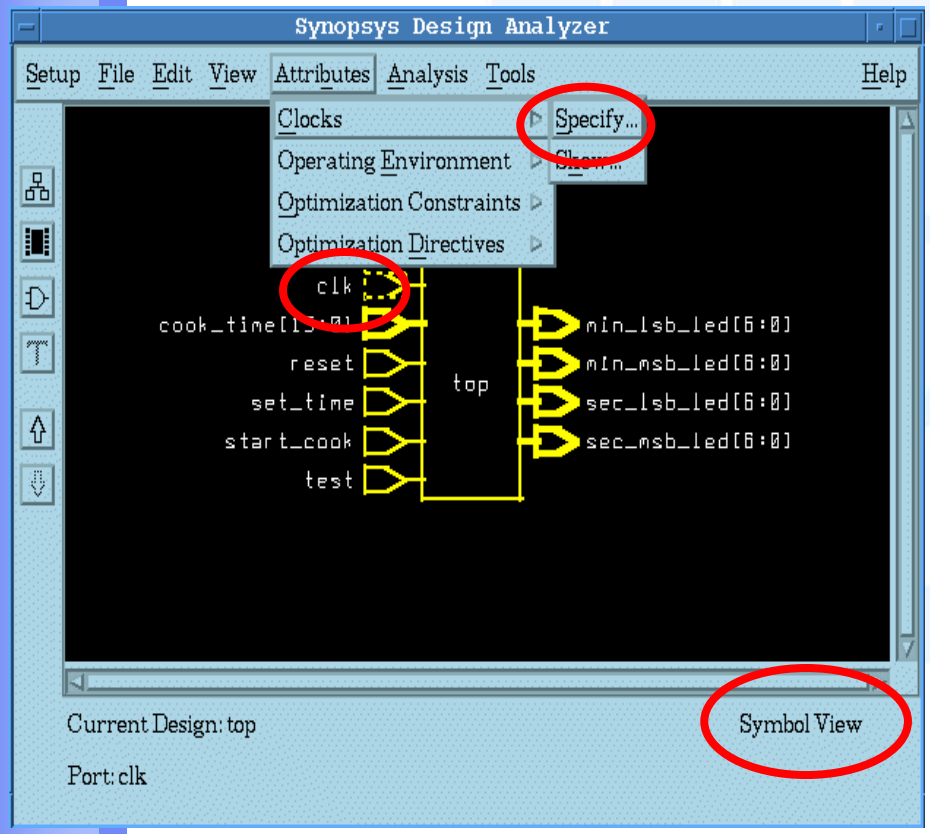
Finite-state-machine design represented as a state table

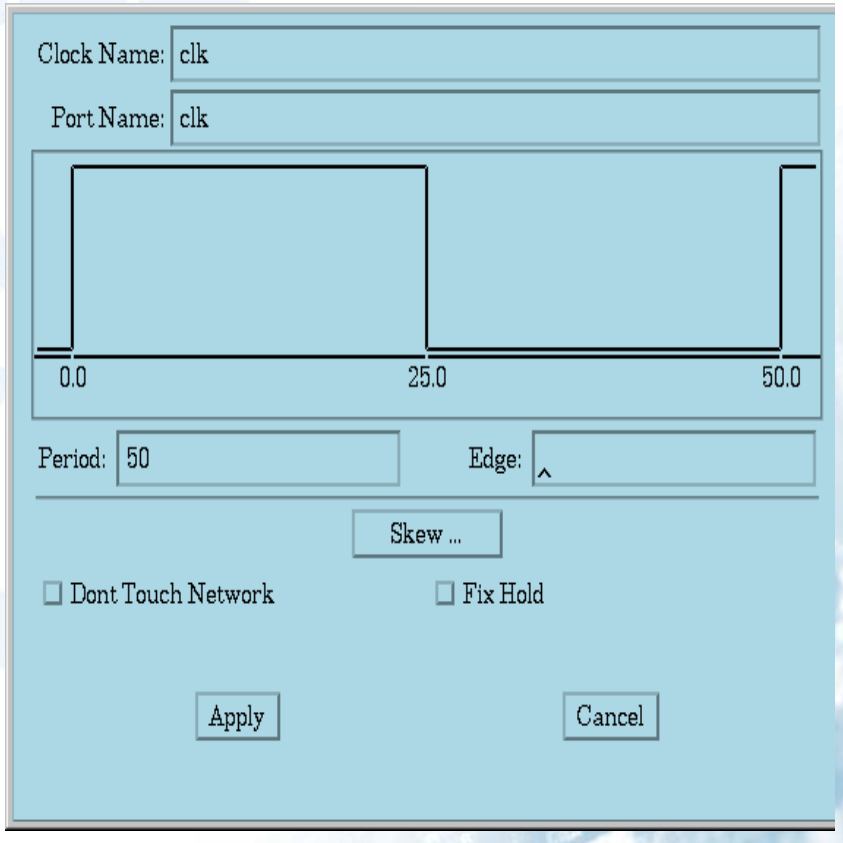
NETLIST

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 Analyzer to constrain your design



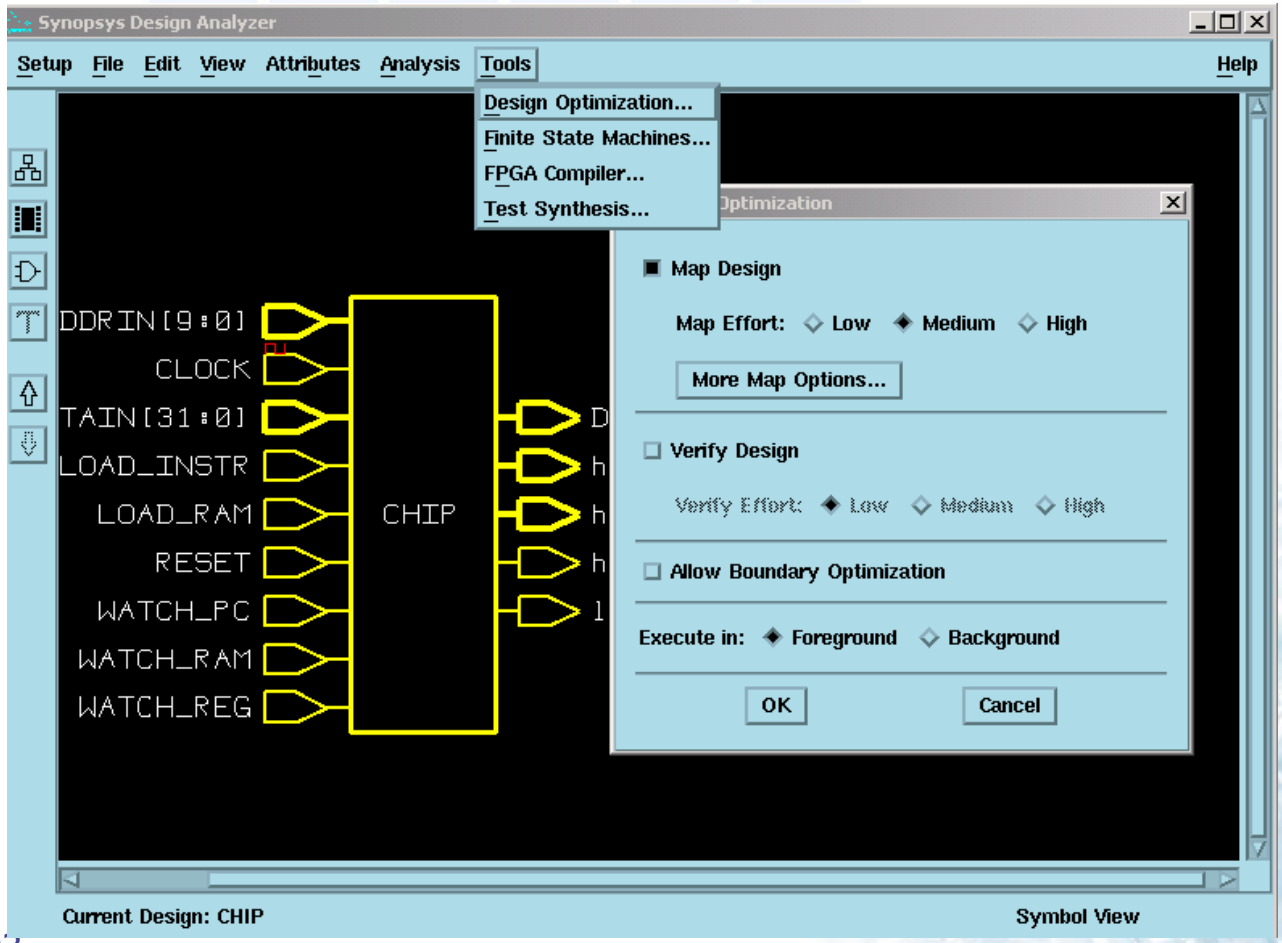


Check Design

- Analysis/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`

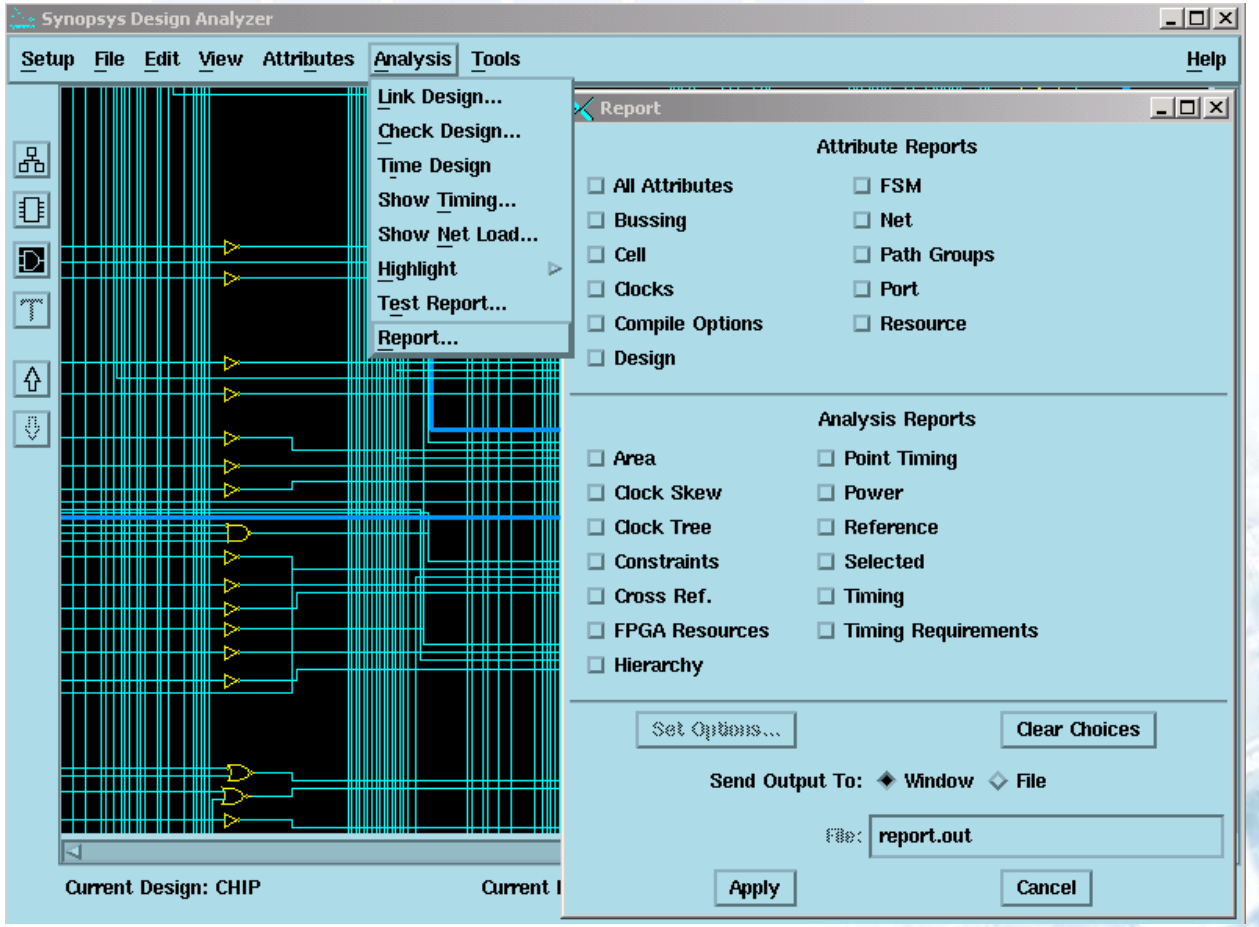
Compile the Design

- The `compile` command optimizes and maps the current_design



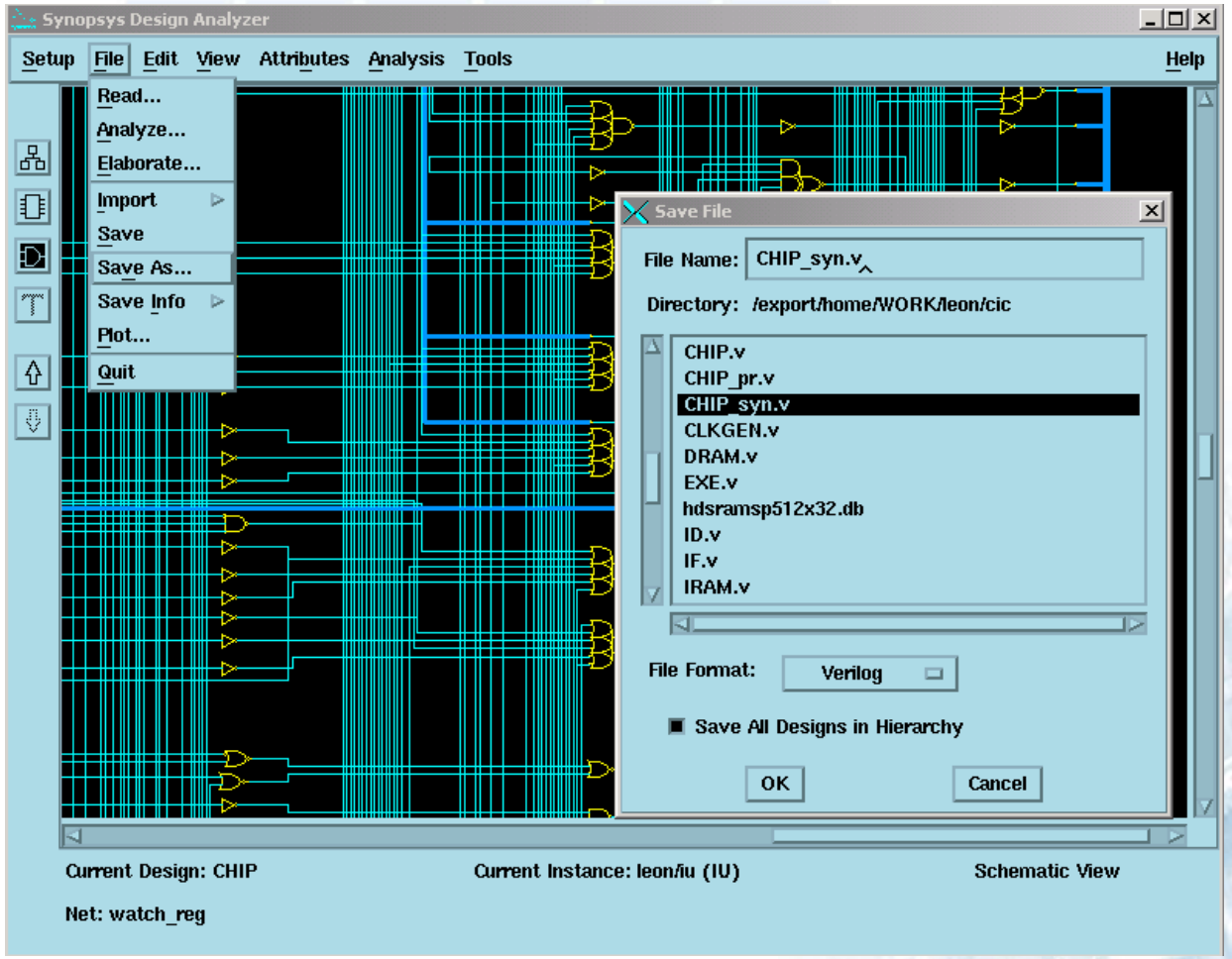
Analyze the Design

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



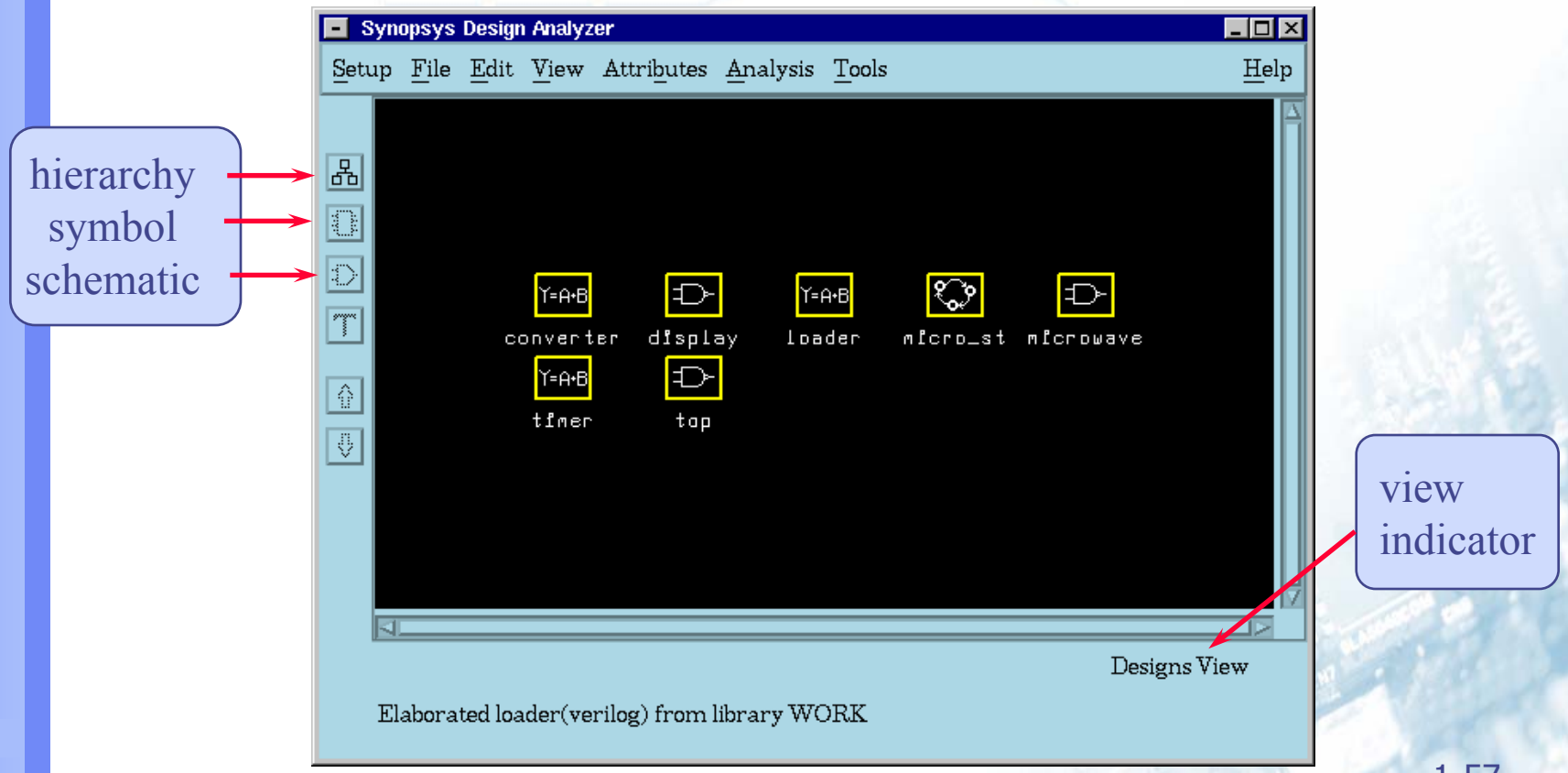
Save the Design

- Write out the design netlist after synthesis



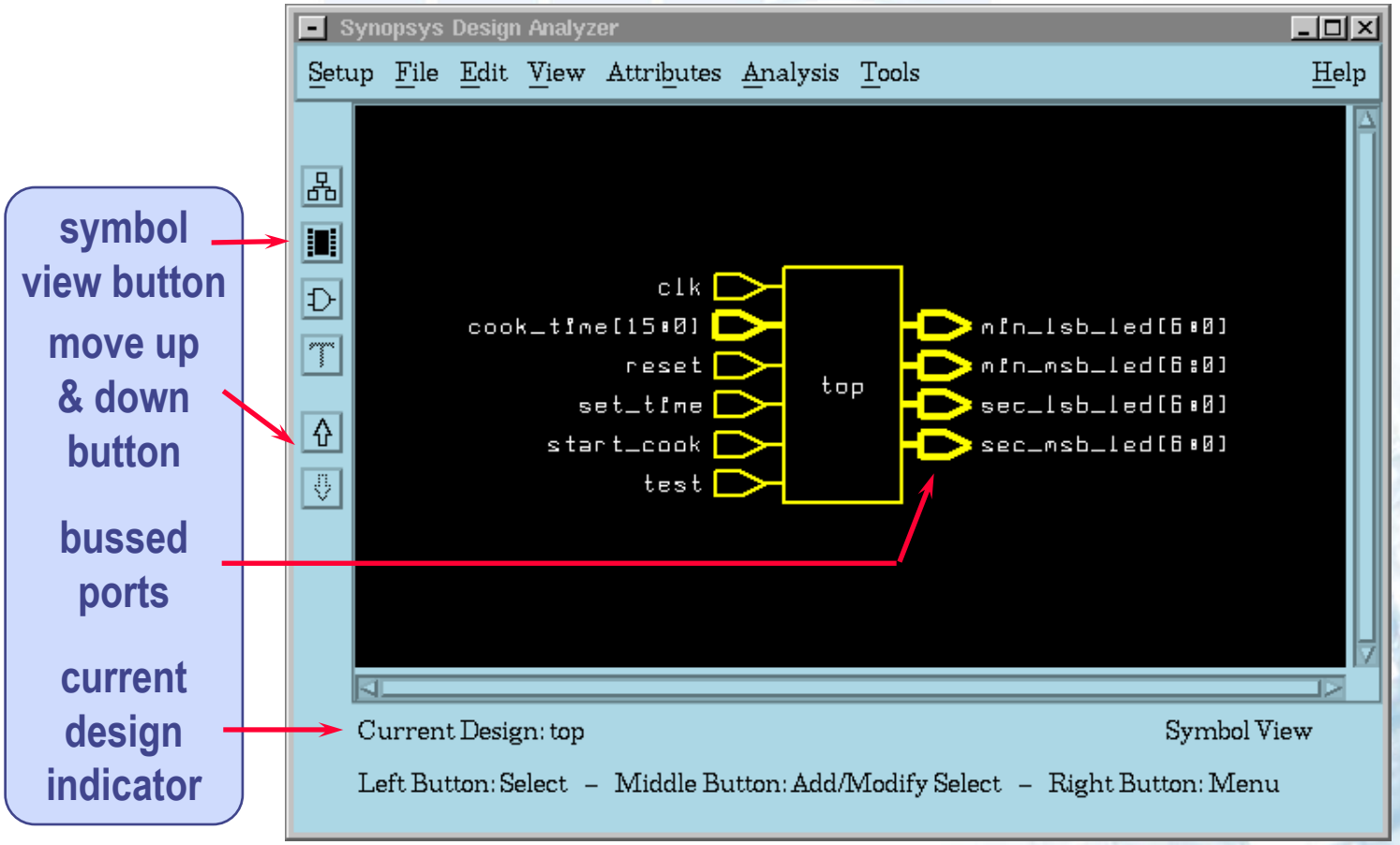
Four Different View - Design View

- To return to design view
  - View/Change Level/Top



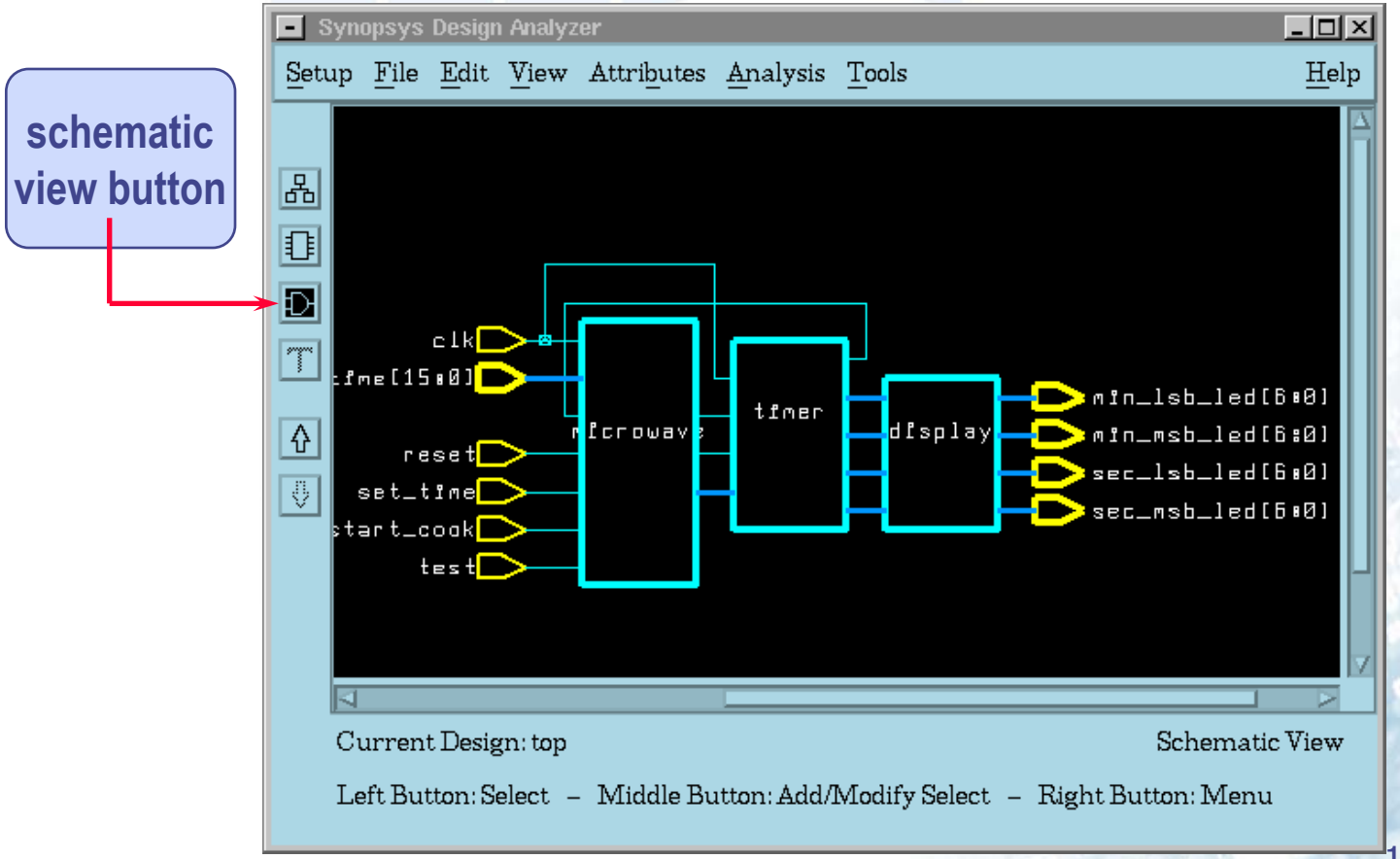
Four Different View - Symbol View

- We usually set port attributes in symbol view



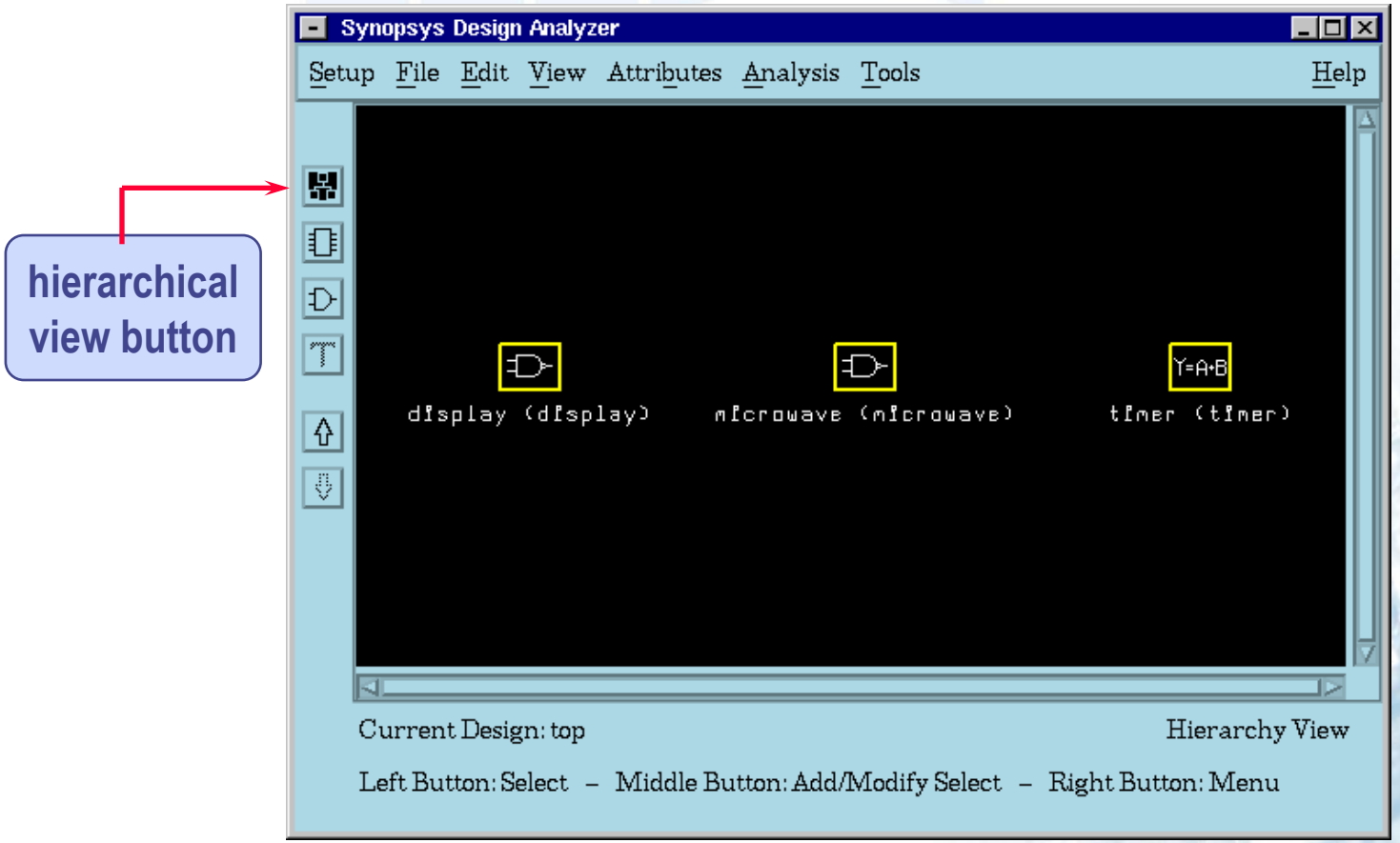
Four Different View - Schematic

- Let you see the schematic view of current design



Four Different View - Hierarchical

- Let you see the hierarchical structure of the current design



Coding Style for Synthesis

❍HDL Coding Style for Synthesis

- ➔Synthesizable Verilog HDL
- ➔Some tricks in Verilog HDL
- ➔Designware library

Synthesizable Verilog HDL

Verilog Module

Module

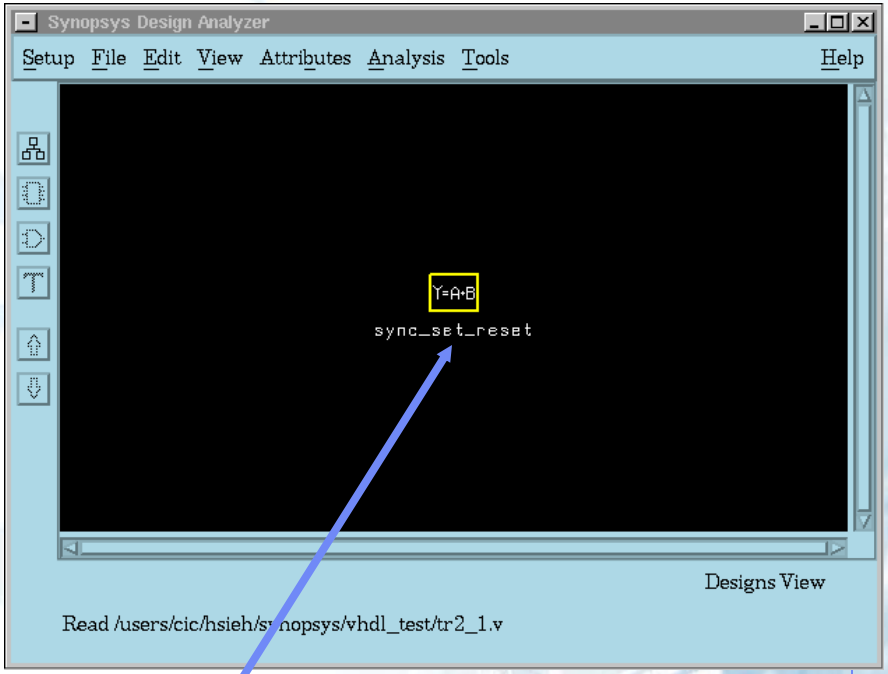
- Module Name & Port List
- Definitions Port,Wire,Register Parameter,Integer,Function
- Module Instantiations
- Module Statements & Constructs

```
module test(a,b,c,d,z,sum);
  input  a,b;          --Inputs to nand gate
  input[3:0] c,d;      --Bused Input
  output z;            --Output from nand gate
  output[3:0] sum;     --Output from adder
  wire   and_out;      --Output from and gate
  reg [3:0] sum;       --Bused Output

  AND instance1(a,b,and_out);
  INV instance2(and_out, z);

  always @(c or d)
     begin
         sum = c + d;
     end
endmodule
```

Verilog Module

- A Verilog module is a Synopsys design block
- Module instantiation gives design hierarchy

```
module sync_set_reset(reset,set,d,gate
,y,t);
input reset,set,gate,d;
output y,t;
//synopsys sync_set_reset "reset,set"
reg y,t;
always @(reset or set)
begin : direct_set_reset
if (reset)
y=1'b0;
else if (set)
y=1'b1;
end
endmodule
```



Design block

Verilog Basis & Primitive Cell (1/2)

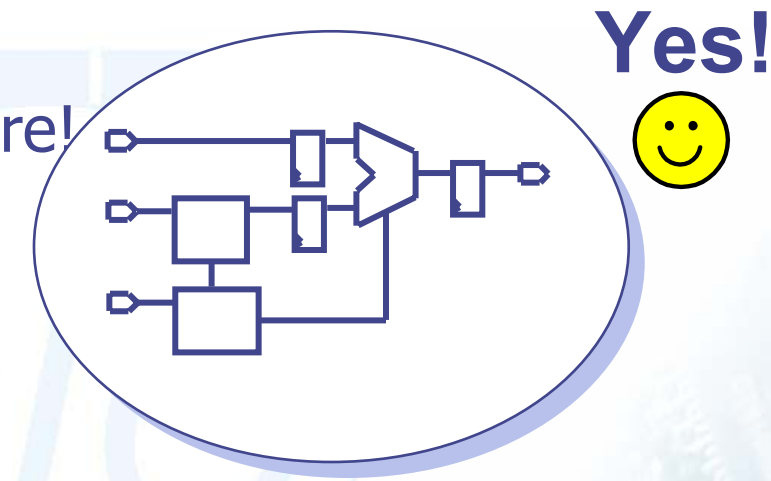
- Verilog Basis
  - parameter declarations
  - wire, wand, wor, tri, supply0, and supply1 declarations
  - reg declarations
  - input declarations
  - output declarations
  - inout declarations
  - Continuous assignments
  - Module instantiations
  - Gate instantiations
  - Function definitions
  - always blocks
  - task statements

Verilog Basis & Primitive Cell (2/2)

- Verilog primitive cells build basic combinational circuit
- Synthesizable Verilog primitives cells
  - and, nand, or, nor, xor, xnor, not
  - bufif0, bufif1, notif0, notif1

The 3 BIG Picture Guidelines (1/3) - Think of Hardware

- Write the Register Transfer Language.
- When you use the HDL, ***ALWAYS*** think of hardware!



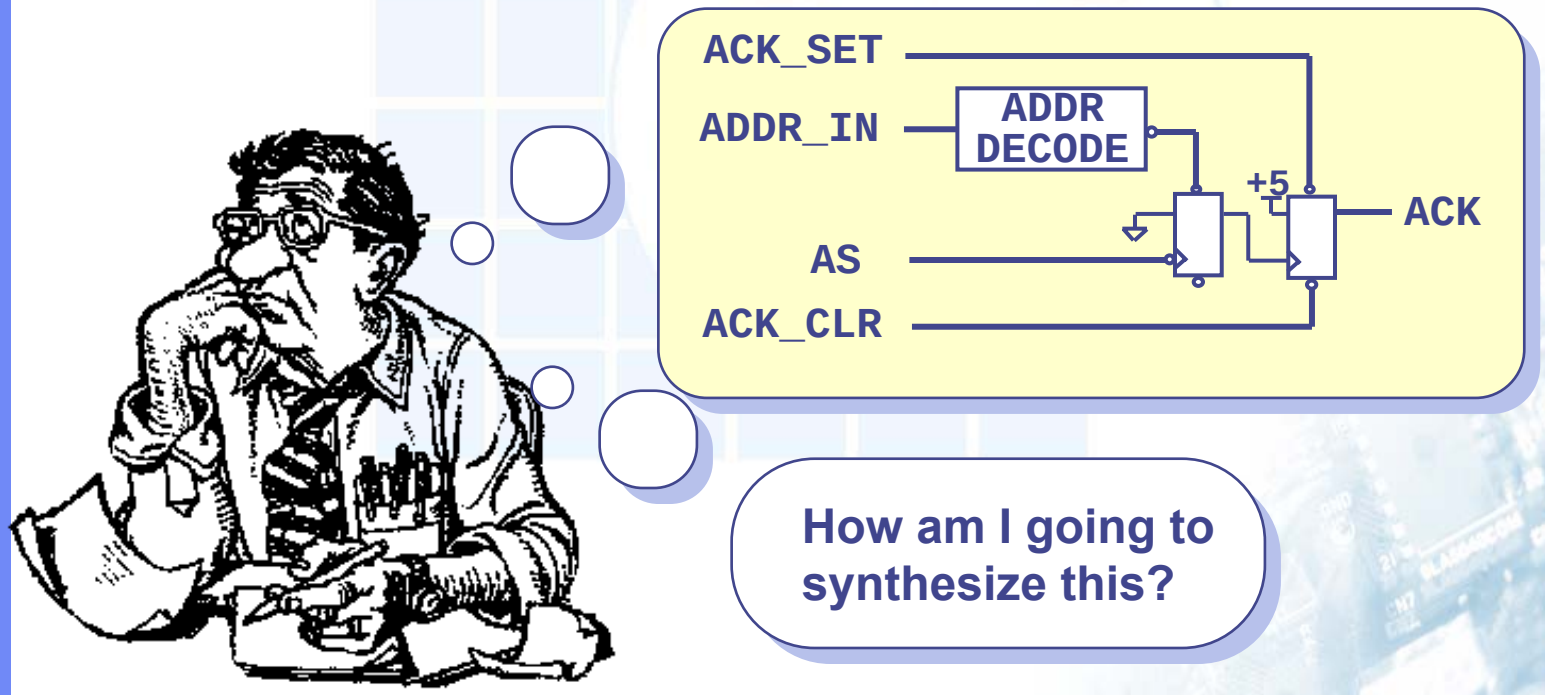
Yes!

```
after 20 ns and
    2 clock cycles
    OUTPUT <= IN1 + RAM1;
    wait 20 ns;
    ...
```

No!

The 3 BIG Picture Guidelines (2/3) - Think of Synchronous Hardware

- Synchronous design can run smoothly during synthesis, test , simulation and layout.
- Asynchronous design should be avoided as possible!



The 3 BIG Picture Guidelines (3/3) - Think RTL

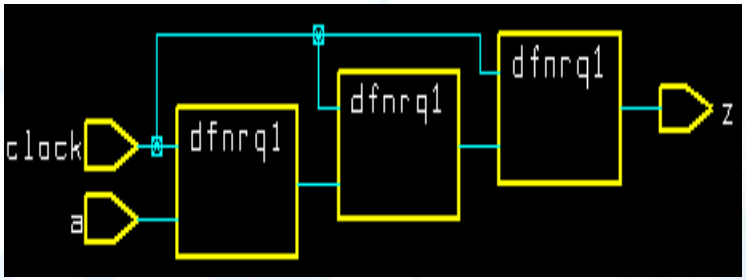
- RTL = Register Transfer Level
  - Writing in an RTL coding style means describing
    - the register architecture,
    - the circuit topology, and
    - the functionality between registers
  - Design Compiler optimizes logic between registers
    - It does not optimize the register placement

Non-Blocking and Blocking

- For the most straightforward code:
  (you get what you expect)
  - Use *non-blocking* assignments within sequential always block.
  - Example:

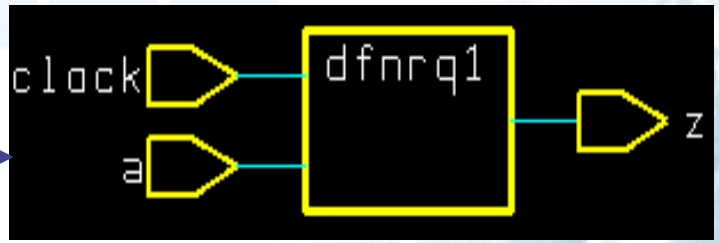
```
always @(posedge clock) begin
      x <= a;
      y <= x;
      z <= y;
end
```

Usually you expect




```
always @(posedge clock) begin
      x = a;
      y = x;
      z = y;
end
```

May not you expect

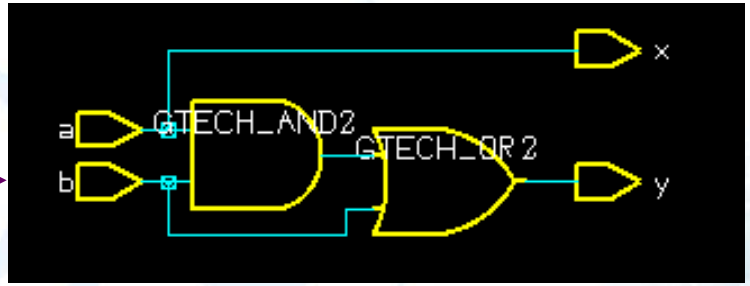


Non-Blocking and Blocking (cont.)

- For the most straightforward code:
  (you get what you expect)
  - Use *blocking* assignments within combinational always block.
  - Example:

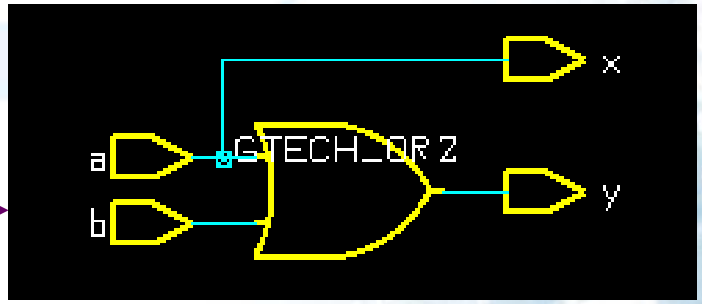
```
always @(a or b or x) begin
    x = a & b;
    y = x | b;
    x = a;
end
```

Usually you expect




```
always @(a or b or x) begin
    x <= a & b;
    y <= x | b;
    x <= a;
end
```

May not you expect



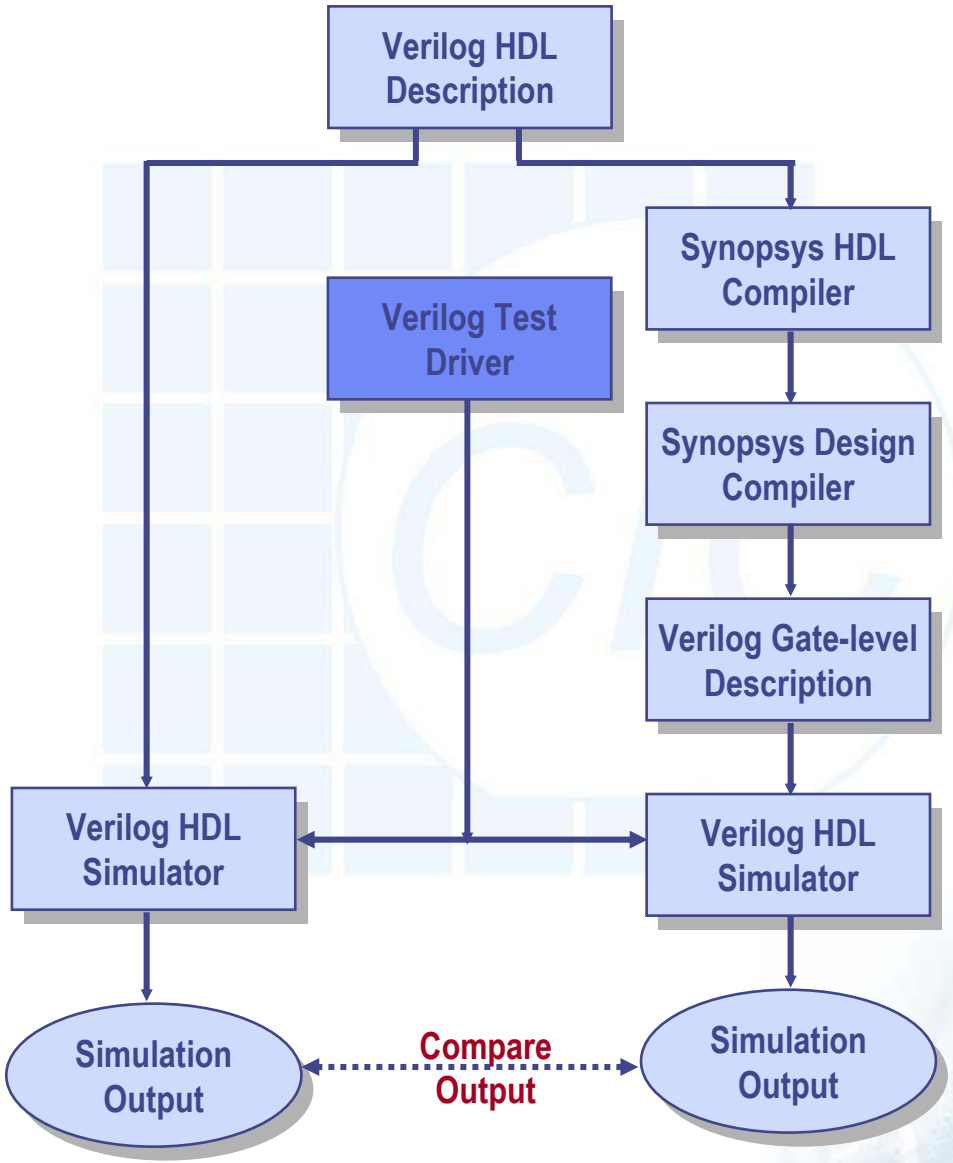
Synthesizable Verilog Code

- Synopsys can't accept all kinds of Verilog constructs
- Synopsys can only accept a subset of Verilog syntax and this subset is called "**Synthesizable Verilog Code**"
- The same situation also exists in VHDL
- This chapter will introduce synthesizable verilog coding style to you, and this is the first challenge when you use Synopsys to convert your RTL code to Gate level netlist

HDL Compiler Unsupported

- delay
- initial
- repeat
- wait
- fork
- event
- deassign
- force
- release
- primitive -- User defined primitive
- time
- triand, trior, tri1, tri0, trireg
- nmos, pmos, cmos, rnmos, rpmos, rcmos
- pullup, pulldown
- rtran, tranif0, tranif1, rtranif0, rtranif1
- case identity and not identity operators
- Division and modulus operators
  - division can be done using DesignWare instantiation

Design Methodology



Design Flow

1. Write a design description in the Verilog language. This description can be a combination of structural and functional elements. This description is used with both the Synopsys HDL Compiler and the Verilog simulator.
2. Provide Verilog-language test drivers for the Verilog HDL simulator. The drivers supply test vectors for simulation and gather output data.
3. Simulate the design by using a Verilog HDL simulator. Verify that the description is correct.
4. Synthesize the HDL description with HDL Compiler. HDL Compiler performs architectural optimizations, then creates an internal representation of the design.

Design Flow

5. Use Synopsys Design Compiler to produce an optimized gate-level description in the target ASIC library. You can optimize the generated circuits to meet the timing & area constraints wanted.
6. Use Synopsys Design Compiler to output a gate-level Verilog description. This netlist-style description uses ASIC components as the leaf-level cells of the design. The gate-level description has the same port and module definitions as the original high-level Verilog description.
7. Use the original Verilog simulation drivers from Step 2 because module and port definitions are preserved.
8. Compare the output of the gate-level simulation with the output of the original Verilog description simulation to verify that the implementation is correct.

Wire & Reg

- wire(wand, wor, tri)
  - Physical wires in a circuit
  - Cannot assign a value to a wire within a function or a begin…..end block
  - A wire does not store its value, it must be driven by
    - by connecting the wire to the output of a gate or module
    - by assigning a value to the wire in a continuous assignment
  - An un-driven wire defaults to a value of Z (high impedance).
  - Input, output, inout port declaration -- wire data type (default)

Wire & Register

- reg
  - A variable in Verilog
- Use of "reg" data type is not exactly synthesized to a really register.
- Use of wire & reg
  - When use "wire" ➔ usually use "assign" and "assign" **does not** appear in "always" block
  - When use "reg" ➔ only use "a=b" , always appear in "always" block

```
module test(a,b,c,d);
input a,b;
output c,d;
reg d;
assign c=a;
always @(b)
  d=b;
endmodule
```

Continuous Assignment (1/2)

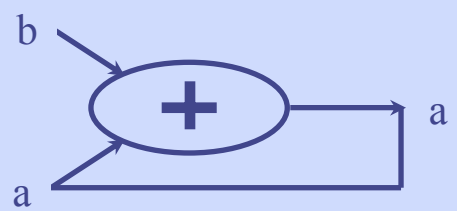
- Drive a value onto a wire, wand, wor, or tri
  - Use an explicit continuous assignment statement after declaration
  - Specify the continuous assignment statement in the same line as the declaration for a wire
- Used for datapath descriptions
- Used to model combinational circuits
- Example

```
wire a;          --declare
assign a=b&c;    --assign

wire a=b&c;      --declare and assign
```

Continuous Assignment (2/2)

- Avoid logic loop
  - HDL Compiler and Design Compiler will automatically open up asynchronous logic loops
  - Without disabling the combinational feedback loop, the static timing analyzer can't resolve
  - Example

```
assign a = b + a ;
```



Verilog Operators Supported

- Binary bit-wise (~,&,|,^,~^)
- Unary reduction (&,~&,|,~|,^,~^)
- Logical (!,&&,||)
- 2‘s complement arithmetic (+,-,*,/,%)
- Relational (>,<,>=,<=)
- Equality (==,!=)
- Logical shift (>>,<<)
- Conditional (?:)

Verilog Operators Supported

Operator	Description
{ }	concatenation
+ - * /	arithmetic
%	modulus
> >= < <=	relational
!	logical NOT
&&	logical AND
| |	logical OR
==	logical equality
! =	logical inequality
~	bit-wise NOT
&	bit-wise AND
|	bit-wise OR
^	bit-wise XOR

Operator	Description
^ ~ ~ ^	bit-wise XNOR
&	reduction AND
|	reduction OR
~ &	reduction NAND
~ |	reduction NOR
^	reduction XOR
~ ^ ^ ~	reduction XNOR
<<	left shift
> >	right shift
?:	conditional

Comparisons to X or Z

- Comparisons to an X or Z are always ignored.
- Comparison is always evaluated to false, which may cause simulation & synthesis mismatch.

```
module compare_x(A,B);
input A;
output B;
reg B;

always begin
  if (A == 1'bx)
    B = 0;
  else
    B = 1;
end

endmodule
```

Warning: Comparisons to a "don't care" are treated as always being false in routine compare_x line 7 in file "compare_x.v" this may cause simulation to disagree with synthesis. (HDL-170)

Bit-wise,Unary,Logical Operator

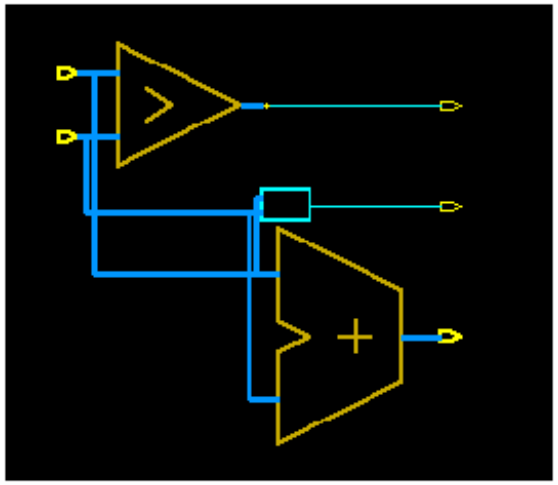
a = 1011
b = 0010

bit-wise	unary reduction	logical
a | b = 1011	| a = 1	a || b = 1
a & b = 0010	& a = 0	a && b = 1

Arithmetic, Relational, Equality

- These operators usually work on vectors.
- Each use of these operators results in utilization of a resource block.

```
wire [7:0] c = a + b;
wire a_bigger = a > b;
wire eq = ( a==b );
```



Resource Blocks

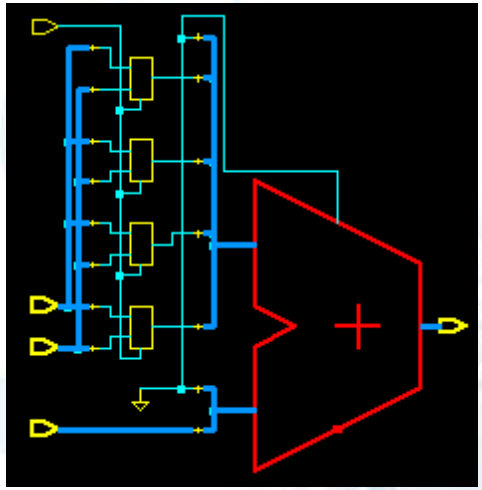
- These blocks are known as DesignWare parts of synthetic cells
- If the operation is > 4 bit, the DesignWare part will become a level of hierarchy, and generate a new design block after design compiled
- If the operation is < or = 4 bit, it won't become a level of hierarchy
- Optimization constraints determine the architecture of DesignWare part, each block can be further optimized for its environment and interface

Some Tricks in Verilog HDL

Resource Sharing

- Operations can be shared if they lie in the same always block.

```
always @(a or b or c or sel)
if (sel)
        z = a + b ;
else
        z = a + c ;
```



Resource Sharing

- Analysis/Report/Resource

Report Output

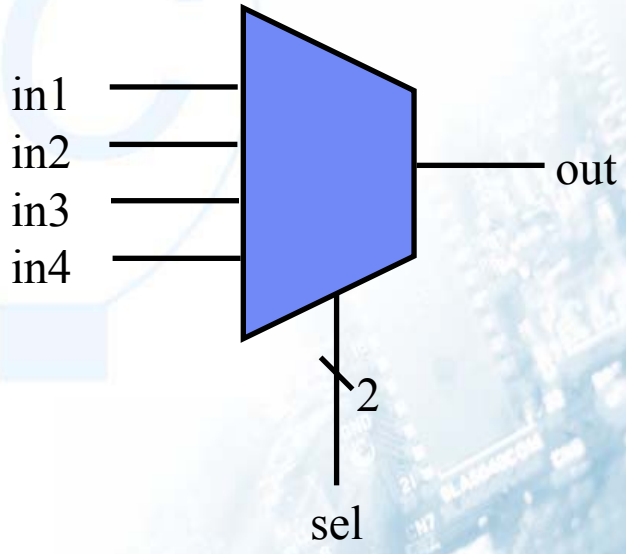
```
design_analyzer>
****************************************
Report : resources
Design : lab1
Version: 2002.05-SP1
Date   : Wed May 14 17:24:53 2003
****************************************

Resource Sharing Report for design lab1 in file
        /users2/cic/ccyang/SYNOPSYS/LAB/lab1/lab1.v
==============================================================================
|          |              |            | Contained     |                     |
| Resource | Module       | Parameters | Resources     | Contained Operations |
==============================================================================
| r236     | DW01_add     | width=9    |               | add_8 add_9         |
==============================================================================

Implementation Report
==========================================================================
|                  |                  | Current          | Set              |
| Cell             | Module           | Implementation   | Implementation   |
==========================================================================
| r236             | DW01_add         | rpl              |                  |
==========================================================================
```

Show Next Previous Cancel

Conditional Operator

- The value assigned to LHS is the one that results TRUE from the expression evaluation.
- This can be used to model multiplexer.
- Can be nested.

```
assign out = (sel == 2'b00 ) ? in1 :
             (sel == 2'b01 ) ? in2 :
             (sel == 2'b10 ) ? in3 :
             (sel == 2'b11 ) ? in4 :
                               1'bx;
```



Concatenation operator

- Combine one or more expressions to form a larger vector. ex. `3'b100 --> {1'b1,{2{1'b0}}}`
- If you want to transfer more than one data from function construct, concatenation operator is a good choice.

```
Output [7:0] ccr;
wire a,b,c,d,e,f;
…
assign ccr = { 2'00,a,b,c,d,e,f };
```

```
output [7:0] ccr;
wire a,b,c,d,e,f;
…
assign ccr[7] = 1'b0 ;
assign ccr[6] = 1'b0 ;
assign ccr[5] = a ;
assign ccr[4] = b ;
assign ccr[3] = c ;
assign ccr[2] = d ;
assign ccr[1] = e ;
assign ccr[0] = f ;
```

```
function [8:0] adder;
input [7:0] a, b;
reg c;
reg [7:0] temp;
integer i;
begin
c = 0;
for (i = 0; i <= 7; i = i + 1) begin
  temp[i] = a[i] ^ b[i] ^ c;
  c = a[i] & b[i] | a[i] & c | b[i] & c;
end
end
adder = { c , temp };
endfunction

assign {cout, sum} = adder(a,b);
```

Function declarations

- Function declarations are one of the two primary methods for describing combinational logic.
- Can be declared and used within a module.
- Function construct

```
function [range] name_of_function ;
  [func_declaration]
  statement_or_null
endfunction
```

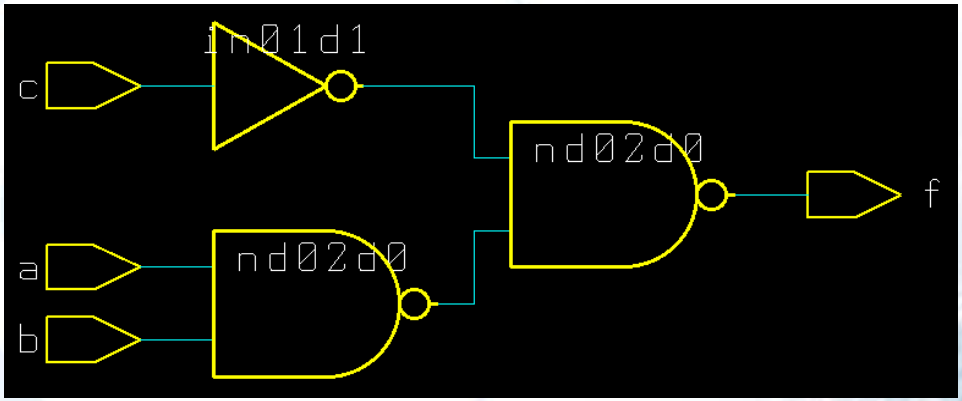
```
function [8:0] adder;
input [7:0] a, b;
reg c;
reg [7:0] temp;
integer i;
begin
c = 0;
for (i = 0; i <= 7; i = i + 1) begin
  temp[i] = a[i] ^ b[i] ^ c;
  c = a[i] & b[i] | a[i] & c | b[i] & c;
end
end
adder = { c , temp};
endfunction

assign {cout, sum} = adder(a,b);
```

Combinational Always Block

- Sensitivity list must be specified completely, otherwise synthesis may mismatch with simulation

```
always @(a or b or c)
f=a&b|c;
```

```
always @(a or b)
f=a&b|c;
```



Warning: Variable 'c' is being read
in routine train line 6 in file '/ccyang/abc/train1.v',
but does not occur in the timing control of the block which begins there. (HDL-180)

`if` Statement (1/4)

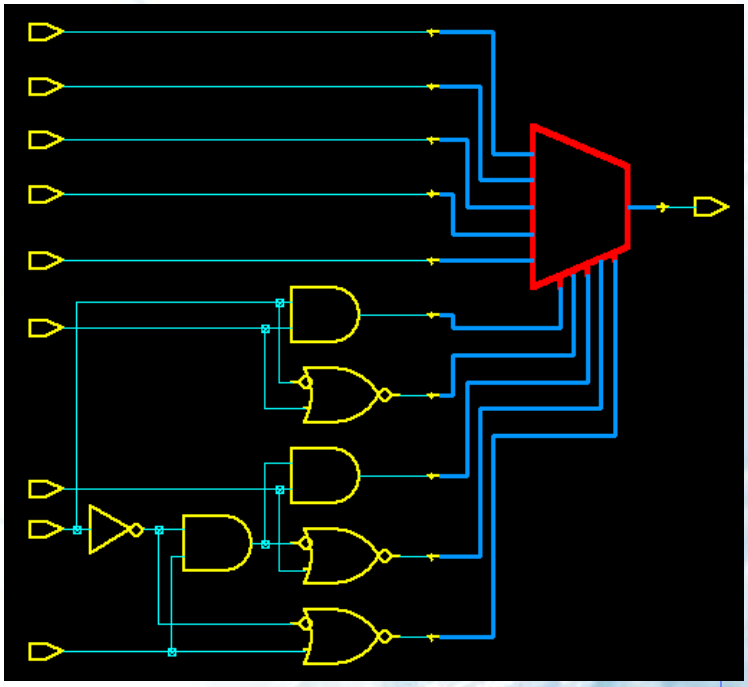
- Provide for more complex conditional actions, each condition expression controls a multiplexer
- legal only in function & always construct
- Syntax

```
if ( expr )
     begin
     ... statements ...
     end
else
     begin
     ... statements ...
     end
```

if Statement (2/4)

- if statement can be nested

```
always @(sel1 or sel2 or sel3 or sel4
or in1 or in2 or in3 or in4 or in5)
begin
if (sel1) begin
  if (sel2) out=in1;
  else out=in2;
end
else if (sel3) begin
  if (sel4) out=in3;
  else out=in4;
end
else out=in5;
end
```



if Statement (3/4)

- What's the difference between these two coding styles?

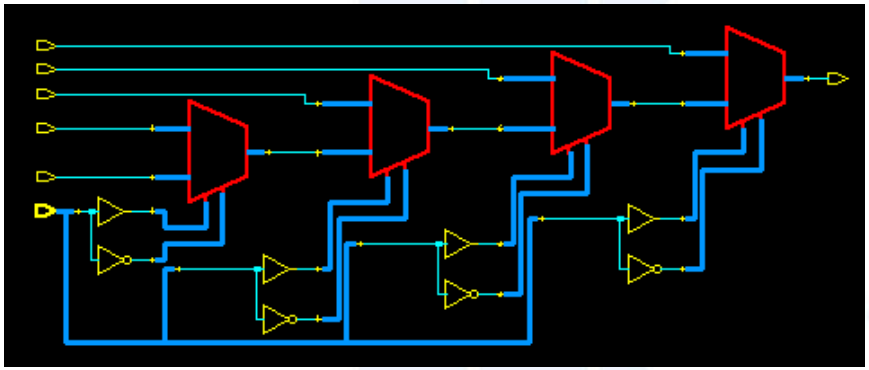
```
module mult_if(a, b, c, d, e, sel, z);
input a, b, c, d, e;
input [3:0] sel;
output z;
reg z;
always @(a or b or c or d or e or sel)
begin
z = e;
if (sel[0]) z = a;
if (sel[1]) z = b;
if (sel[2]) z = c;
if (sel[3]) z = d;
end
endmodule
```

1

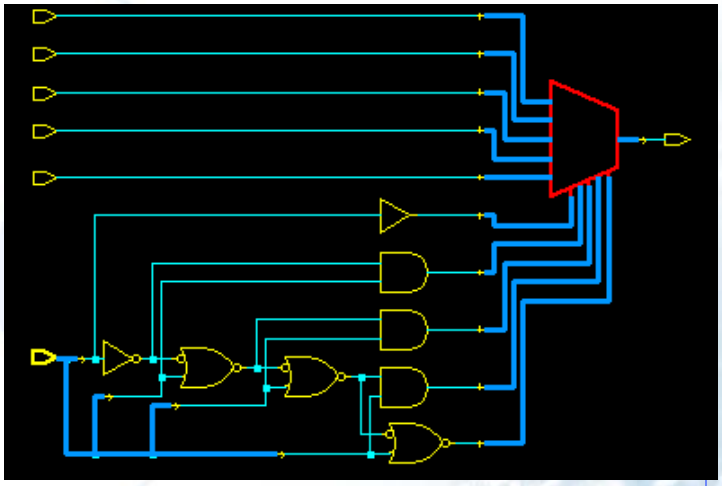
```
module single_if(a, b, c, d, e, sel, z);
input a, b, c, d, e;
input [3:0] sel;
output z;
reg z;
always @(a or b or c or d or e or sel)
begin
z = e;
if (sel[3])
  z = d;
else if (sel[2])
  z = c;
else if (sel[1])
  z = b;
else if(sel[0])
  z = a;
end
endmodule
```

2

`if` Statement (4/4)



1



2

case Statement (1/8)

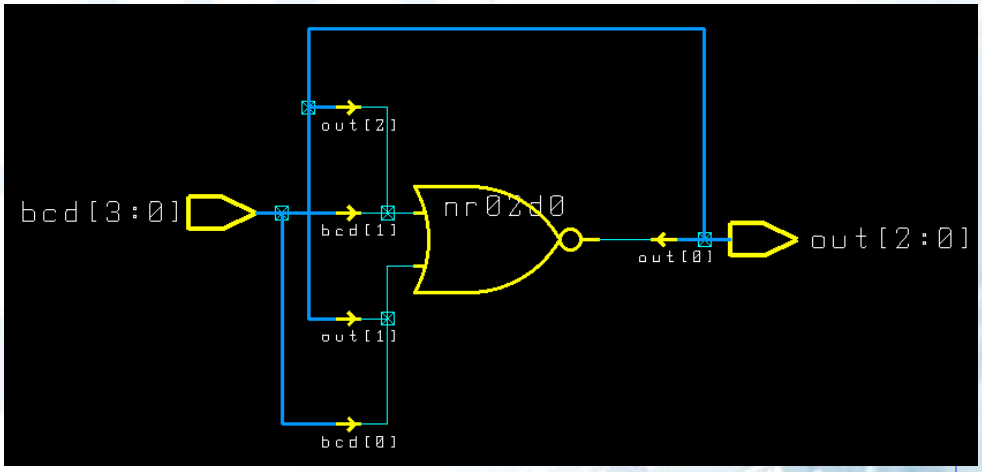
- Legal only in the function & always construct
- syntax

```
case ( expr )
     case_item1: begin
     ... statements ...
     end
     case_item2: begin
     ... statements ...
     end
     default: begin
     ... statements ...
     end
endcase
```

case Statement (2/8)

- A case statement is called a full case if all possible branches are specified.

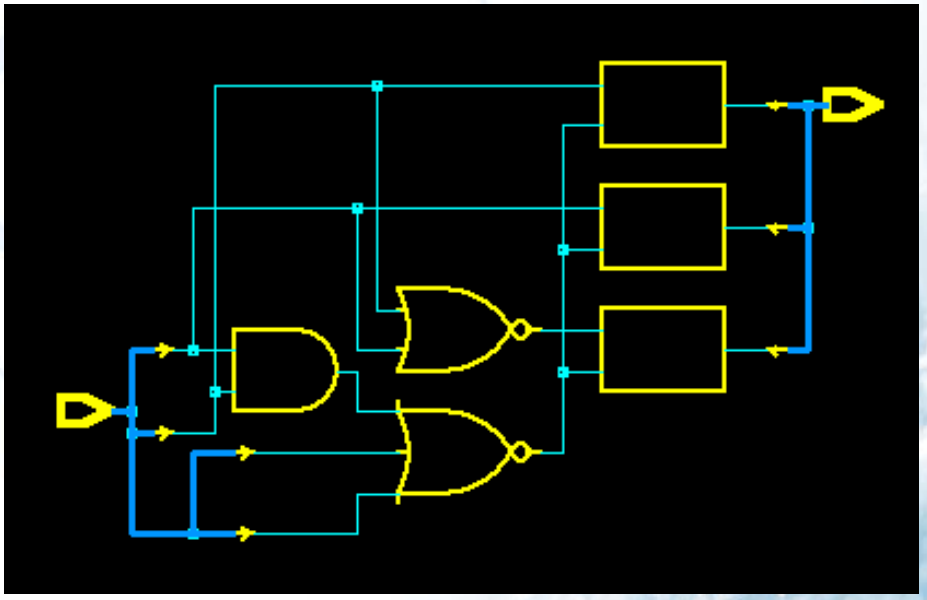
```
always @(bcd) begin
  case (bcd)
        4'd0:out=3'b001;
        4'd1:out=3'b010;
        4'd2:out=3'b100;
        default:out=3'bxxx;
  endcase
end
```



case Statement (3/8)

- If a case statement is not a full case, it will infer a latch.

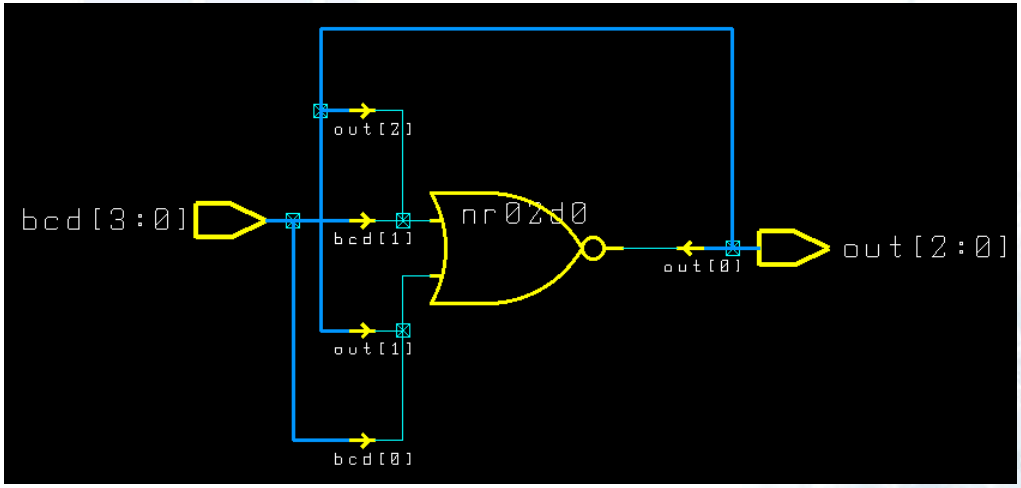
```
always @(bcd) begin
  case (bcd)
        4'd0:out=3'b001;
        4'd1:out=3'b010;
        4'd2:out=3'b100;
  endcase
end
```



case Statement (4/8)

- If you do not specify all possible branches, but you know the other branches will never occur, you can use "//synopsys full_case" directive to specify full case

```
always @(bcd) begin
  case (bcd) //synopsys full_case
        4'd0:out=3'b001;
        4'd1:out=3'b010;
        4'd2:out=3'b100;
  endcase
end
```



case Statement (5/8)

- **Note:** the second case item does not modify reg2, causing it to be inferred as a latch (to retain last value).

```
case (cntr_sig) // synopsys full_case
2'b00 : begin
        reg1 = 0 ;
        reg2 = v_field ;
        end
2'b01 : reg1 = v_field ; /* latch will be inferred for reg2*/

2'b10 : begin
        reg1 = v_field ;
        reg2 = 0 ;
        end
endcase
```

case Statement (6/8)

- Two possible ways we can assign a default value to next_state.

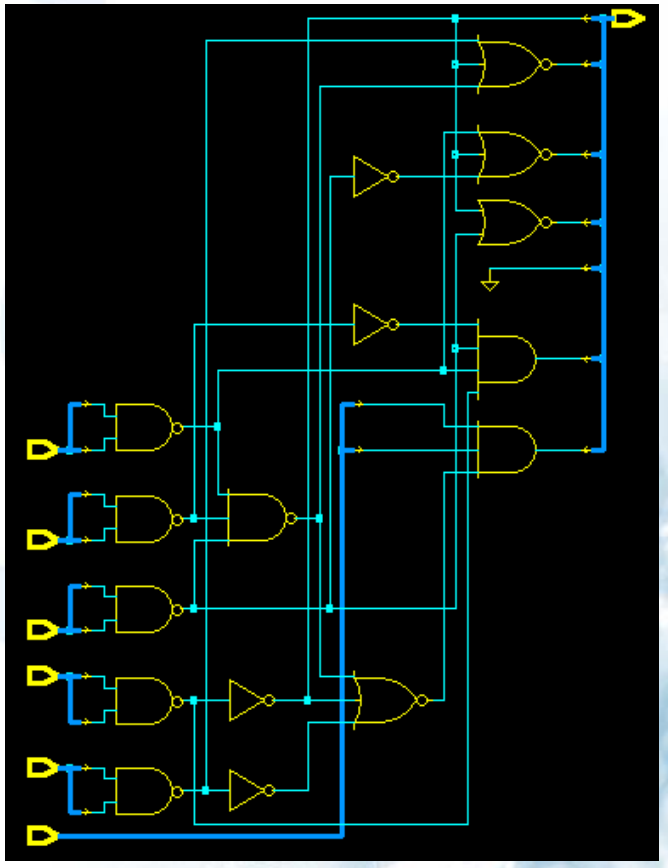
(1)
```
out = 3'b000 ; // this is called unconditional assignment
case (condition)
…
endcase
```

(2)
```
case (condition)
…
default : out = 3'b000 ; // out=0 for all other cases
endcase
```

case Statement (7/8)

- If HDL Compiler can't determine that case branches are parallel, its synthesized hardware will include a priority decoder.

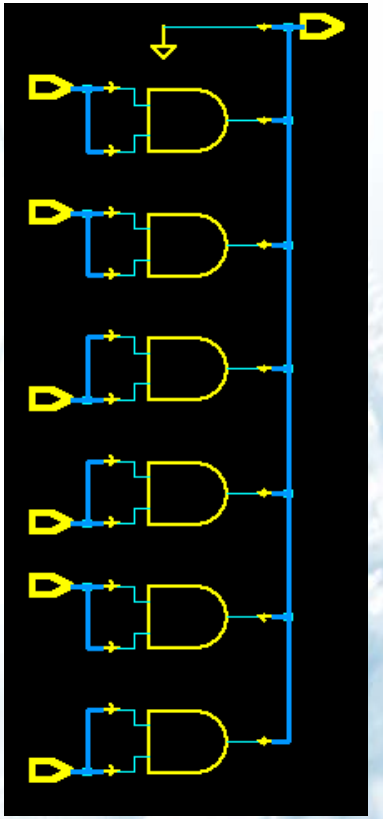
```
always @(u or v or w or x or y or z)
begin
case (2'b11)
  u:out=10'b0000000001;
  v:out=10'b0000000010;
  w:out=10'b0000000100;
  x:out=10'b0000001000;
  y:out=10'b0000010000;
  z:out=10'b0000100000;
  default:out=10'b0000000000;
endcase
end
```



case Statement (8/8)

- You can declare a case statement as parallel case with the "//synopsys parallel_case" directive.

```
always @(u or v or w or x or y or z)
begin
case (2'b11) //synopsys parallel_case
  u:out=10'b0000000001;
  v:out=10'b0000000010;
  w:out=10'b0000000100;
  x:out=10'b0000001000;
  y:out=10'b0000010000;
  z:out=10'b0000100000;
  default:out=10'b0000000000;
endcase
end
```



`if` vs. `case`

- Assume:
  - Assume four mutually exclusive input conditions (X , Y , Z , none of the three)
  - Depending upon which condition is true, your circuit will output a different result.

if/else Solutions

```
always @(X or Y or Z) begin
  out = result1;
  if (X) value = result2;
  if (Y) value = result3;
  if (Z) value = result4;
end
```

```
always @(X or Y or Z) begin
  if (Z) value = result4;
  else if (Y) value = result3;
  else if (X) value = result2;
  else value = result1;
end
```

priority encoded

case Solutions

```
always @(X or Y or Z) begin
  tmp = {X,Y,Z};
  casex(tmp) //synopsys parallel_case
        3`b1xx: value = result2;
        3`bx1x: value = result3;
        3`bxx1: value = result4;
        default: value = result1;
  endcase
end
```

no priority encoding because of **//synopsys parallel_case** directive is used

```
always @(X or Y or Z)
case(1`b1) //synopsys parallel_case
        X: value = result2;
        Y: value = result3;
        Z: value = result4;
        default: value = result1;
endcase
```

one bit comparison

`if` vs. `case`

- Recommendations:
  - If the "if else" chain is too long, use "case" statement to replace them.
  - If you can know the conditions of "case" statement are mutually exclusive, please use synopsys directive "`//synopsys parallel_case`" in order to let design compiler to create a parallel decoder for you.
  - If you know the conditions of a "case" statement, which is not listed, will never occur, please use "`//synopsys full_case`" directive in order to prevent latches been synthesized.

for Loop

- Provide a shorthand way of writing a series of statements.
- Loop index variables must be integer type.
- Step, start & end value must be constant.
- In synthesis, for loops loops are "unrolled", and then synthesized.
- Example

```
always @(a or b) begin
  for( k=0; k<=3; k=k+1 ) begin
    out[k]=a[k]^b[k];
    c=(a[k]|b[k])&c;
  end
end
```

→

```
out[0] = a[0]^b[0];
out[1] = a[1]^b[1];
out[2] = a[2]^b[2];
out[3] = a[3]^b[3];
c = (a[0] | b[0]) & (a[1] | b[1]) &
     a[2] | b[2]) & (a[3] | b[3]) & c;
```

always Block

- Example

```
always @ (event-expression ) begin
  statements
end
```

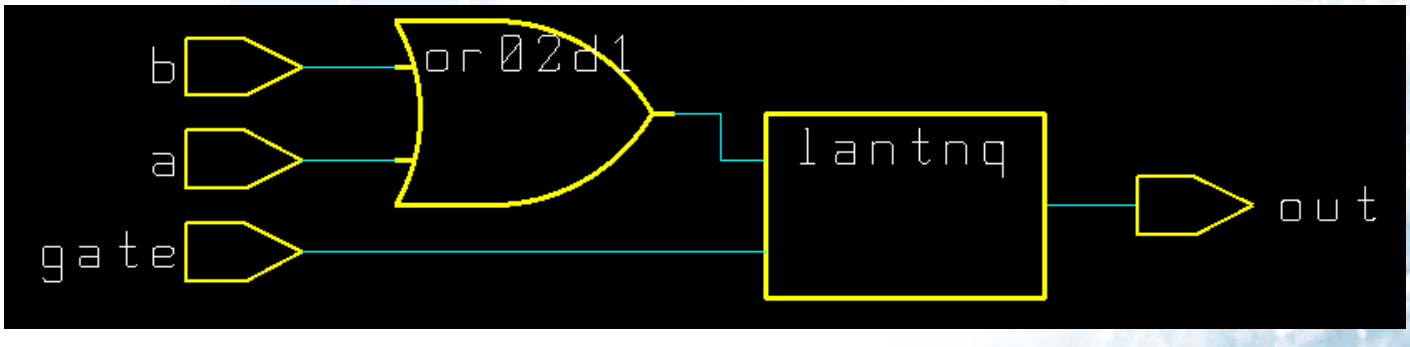
- If event-expression contains posedge or negedge, flip-flop will be synthesized.
- A variable assigned within an always @ block that is not fully specified will result in latches synthesized.
- In all other cases, combinational logic will be synthesized.

Latch Inference

- A variable assigned within an always block that is not fully specified.

```
always @(a or b or gate) begin
  if (gate)
      out = a | b ;
end
```

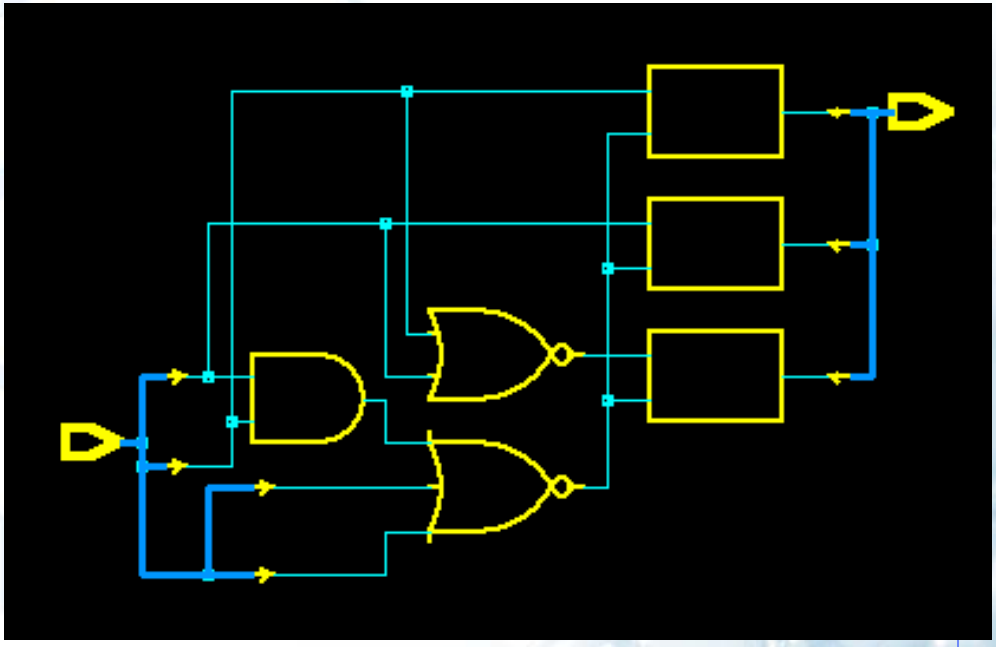
The conditional expression becomes the latch enable



Latch Inference with `case`

- If a case statement is not a full case, it will infer a latch.

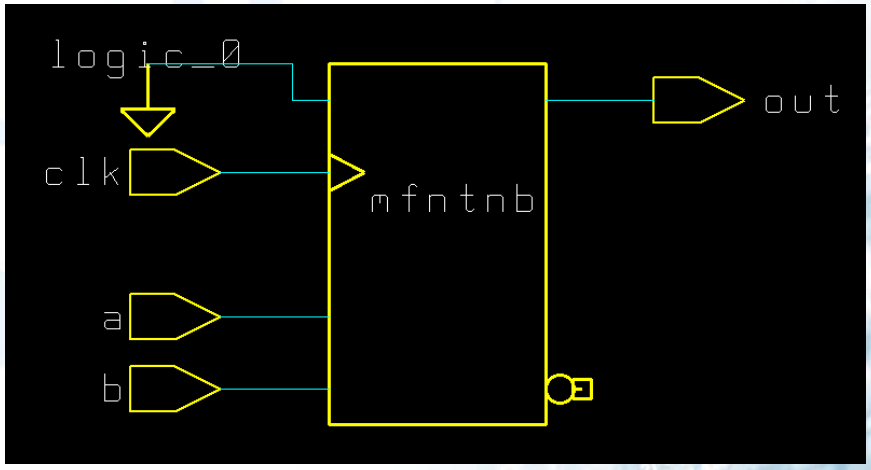
```
always @(bcd) begin
  case (bcd)
        4'd0:out=3'b001;
        4'd1:out=3'b010;
        4'd2:out=3'b100;
  endcase
end
```



Register Inference (1/2)

- A register (flip-flop) is implied when you use the @(posedge clk) or @(negedge clk) in an always block.
- Any variable that is assigned a value in this always block is synthesized as a D-type edge-triggered flip-flop.

```
always @(posedeg clk)
  out <= a & b ;
```

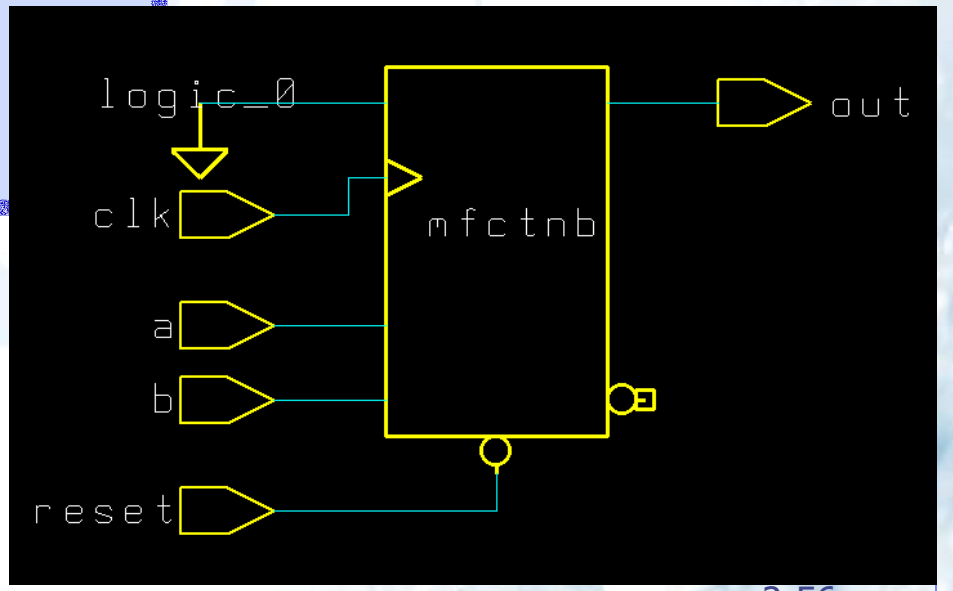


Register Inference (2/2)

- Inference of positive edge clock with active low asynchronous reset flip-flops

```
always @(posedeg clk or negedge reset)
begin
if (!reset)
  out <= 0;
else
  out <= a & b ;
end
```

clocking specified in else

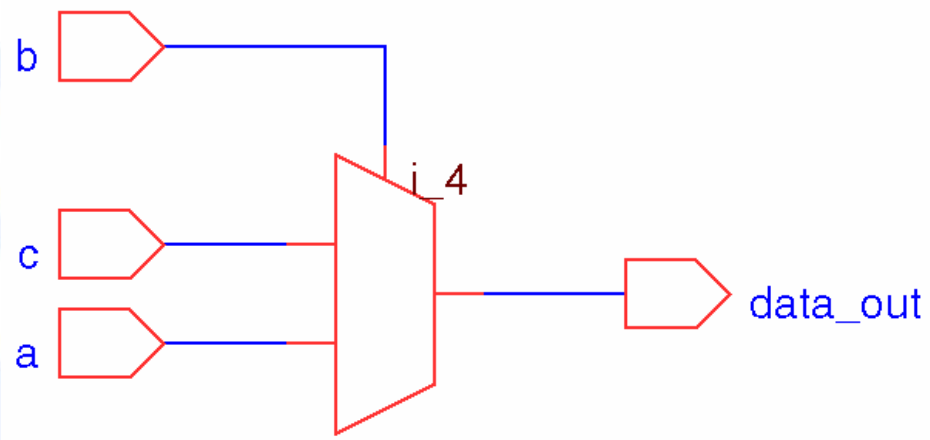


Note: if in always block's sensitive list, you use edge trigger signal, then, **all** signals in this sensitive list must be edge trigger form

Combinational Logic Inference

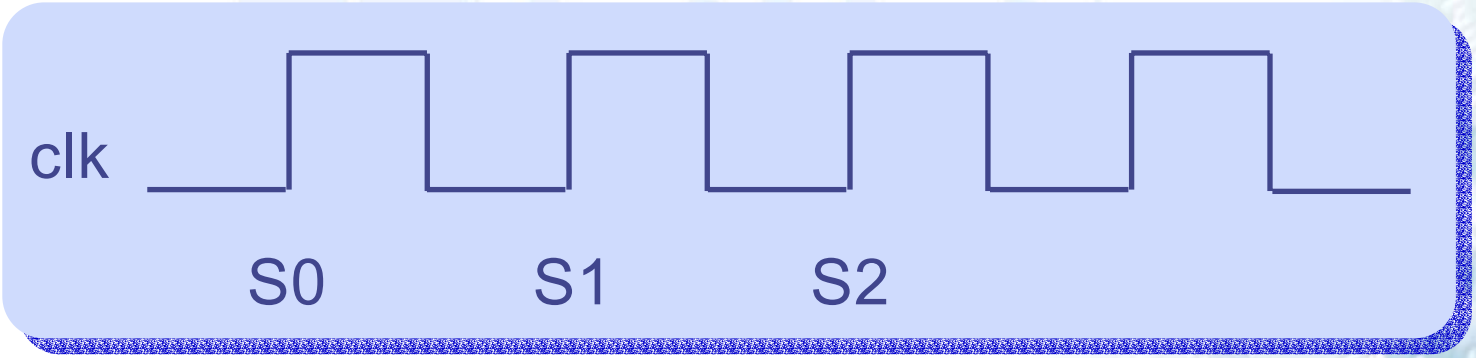
- Not all variables declared as reg data type need a flip-flop or a latch for logic implementation.

```
reg data_out;
always @(a or b or c)
if(b)
    data_out = a;
else
    data_out = c;
```



Implicit Finite State Machine (1/2)

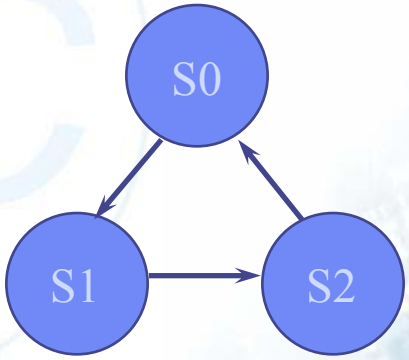
- You can describe a FSM implicitly without define a state register.
- Each clock represents a transition to another state.



Implicit Finite State Machine (2/2)

- Use the implicit state style to describe a single flow of control through a circuit.
- Use single phase clock.

```
always begin
  @(posedge clk)
        total <= data;
  @(posedge clk)
        total <= total + data;
  @(posedge clk)
        total  <= total + data;
end
```



Note: in the same always block, <u>only one type</u> of edge trigger signal can be accepted

Explicit Finite State Machine

- Use explicit FSM to describe asynchronous reset FSM.
- Use if or case statement to allow compact description of state machine logic.

```
always @(current_state or data1 or data2) begin
  case (current_state)
  S0: begin
      result = data1;
      next_state = S1;
      end
  S1: begin
     :
  endcase
end
```

use 1st always block to describe combinational logic

```
always @(posedge clk or negedge reset) begin
  if (!reset)
    current_state <= S0;
  else
    current_state <= next_state;
end
```

use 2nd always block to synthesize state vector, to describe state transition

Finite State Machine Directive

- `//synopsys enum enum_name`
  - Use with Verilog parameter state to specify state machine encoding and where they are used.
- `//synopsys state_vector vector_name`
  - Indicate which variable is chosen as a state vector.

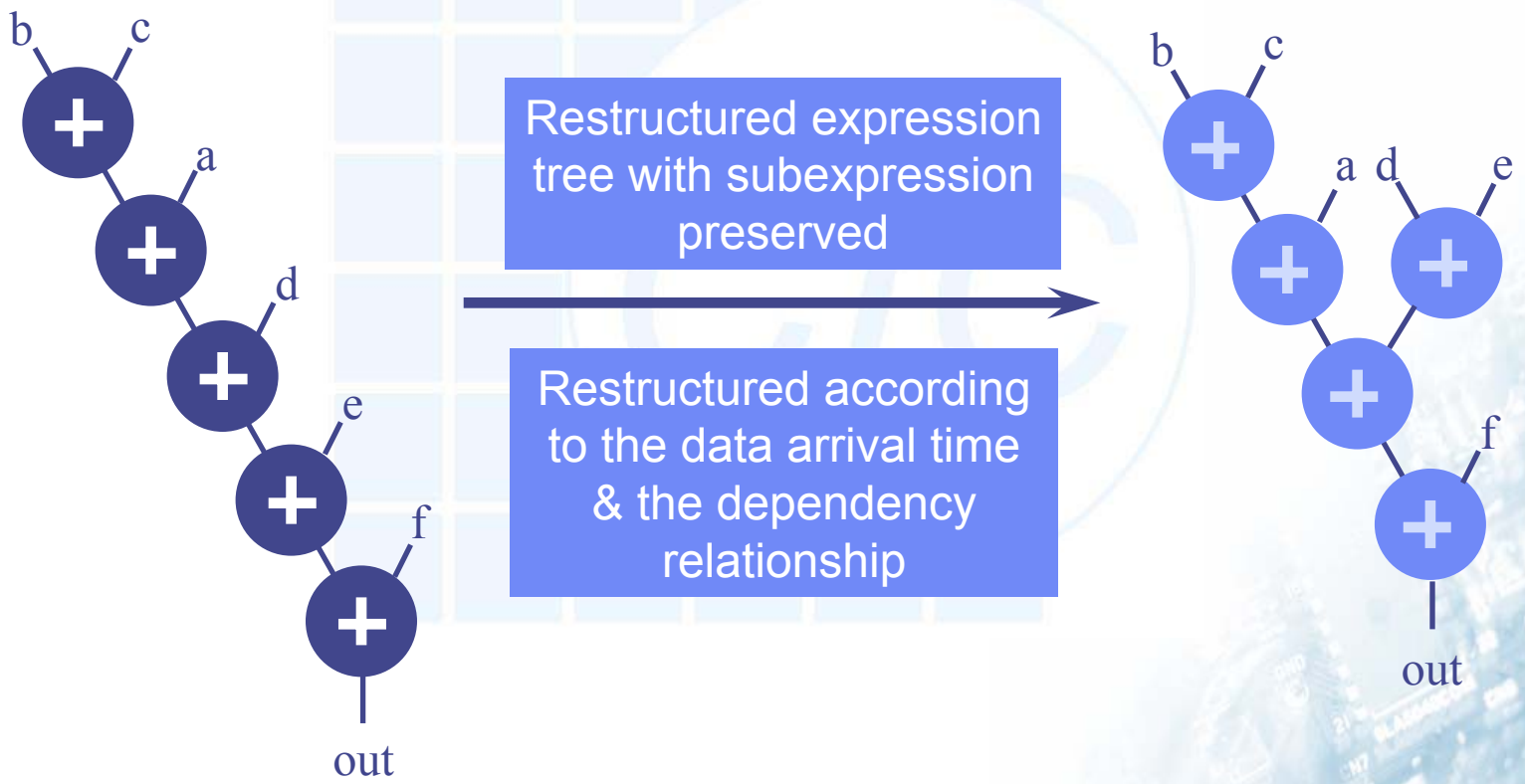
```
/* Define states and encodings */
parameter [2:0] // synopsys enum code
          Grant_A=3'b001, Wait_A=3'b011, Timeout_A1=3'b111,
          Grant_B=3'b010, Wait_B=3'b110, Timeout_B1=3'b101;

reg [2:0] /* synopsys enum code */ present_State, next_State;
//synopsys state_vector present_State
reg ACKA, ACKB, TIMESTART;
always @ (REQA or REQB or TIMEUP  or reset or present_State)
begin
.. .. ..
```

Write Efficient HDL Code

- Use parentheses control complex structure of a design.
- Use operator bit-width efficiently.
- Propagate constant value.

Use Parentheses Properly

- out = ((a+(b+c))+d+e)+f;



Use Operator Bit-Width Efficiently

```
module test(a,b,out);
input [7:0] a,b;
output [8:0] out;
assign out=add_lt_10(a,b);

function [8:0] add_lt_10;
input [7:0] a,b;
reg [7:0] temp;
begin
  if (b<10) temp=b;
  else temp=10;
  add_lt_10=a+temp[3:0]; //use [3:0] for temp
end
endfunction

endmodule
```

Propagate Constant Value

```
parameter size = 8;
wire  [3:0] a,b,c,d,e;
assign c = size + 2;   // constant
assign d = a + 1;      // incrementer
assign e = a + b;      // adder
```

Synopsys HDL Compiler Directive

- What we have mentioned
  - `//synopsys full_case`
  - `//synopsys parallel_case`
  - `//synopsys state_vector vector_name`
  - `//synopsys enum enum_name`
- `//synopsys translate_on` & `//synopsys translate_off` control the HDL Compiler translation of Verilog code off & on
- Your dc_shell script should only contain commands that set constraints & attributes

```
module trivial(a,b,f);
input a,b;
output f;
assign f = a & b;

//synopsys translate_off
initial $monitor(a,b,f);
//synopsys translate_on
endmodule
```

```
//synopsys dc_script_begin
//max_area 50
//set_drive -rise 1 port_b
.........
//synopsys dc_script_end
```

Coding Skill-Data-path Duplication(1/2)

No_dulpicated

```
module BEFORE (ADDRESS, PTR1, PTR2, B, CONTROL, COUNT);
input [7:0] PTR1,PTR2;
input [15:0] ADDRESS, B;
input CONTROL;        // CONTROL is late arriving
output [15:0] COUNT;
parameter [7:0] BASE = 8'b10000000;
wire [7:0] PTR, OFFSET;
wire [15:0] ADDR;
assign PTR = (CONTROL == 1'b1) ? PTR1 : PTR2;
assign OFFSET = BASE - PTR; //Could be any function f(BASE,PTR)
assign ADDR = ADDRESS - {8'h00, OFFSET};
assign COUNT = ADDR + B;
endmodule
```

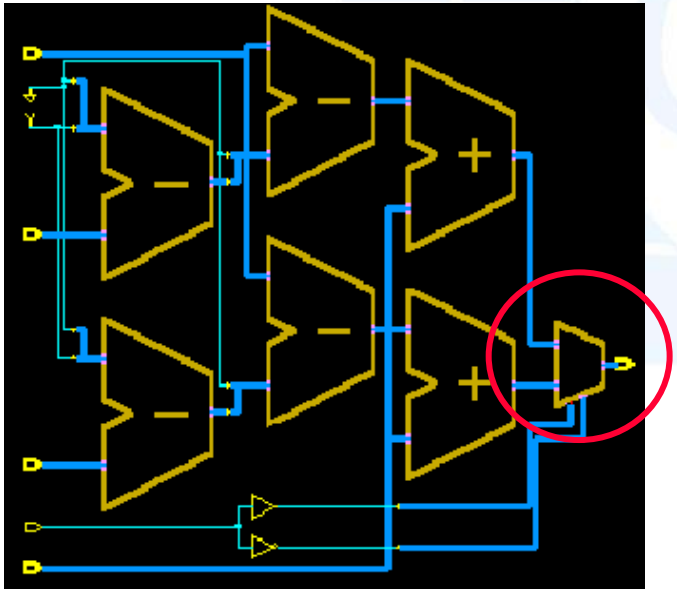
Dulpicated

```
module PRECOMPUTED (ADDRESS, PTR1, PTR2, B, CONTROL, COUNT);
input [7:0] PTR1, PTR2;
input [15:0] ADDRESS, B;
input CONTROL;
output [15:0] COUNT;
parameter [7:0] BASE = 8'b10000000;
wire [7:0] OFFSET1,OFFSET2;
wire [15:0] ADDR1,ADDR2,COUNT1,COUNT2;
assign OFFSET1 = BASE - PTR1;  // Could be f(BASE,PTR)
assign OFFSET2 = BASE - PTR2;  // Could be f(BASE,PTR)
assign ADDR1 = ADDRESS - {8'h00 , OFFSET1};
assign ADDR2 = ADDRESS - {8'h00 , OFFSET2};
assign COUNT1 = ADDR1 + B;
assign COUNT2 = ADDR2 + B;
assign COUNT = (CONTROL == 1'b1) ? COUNT1 : COUNT2;
endmodule
```

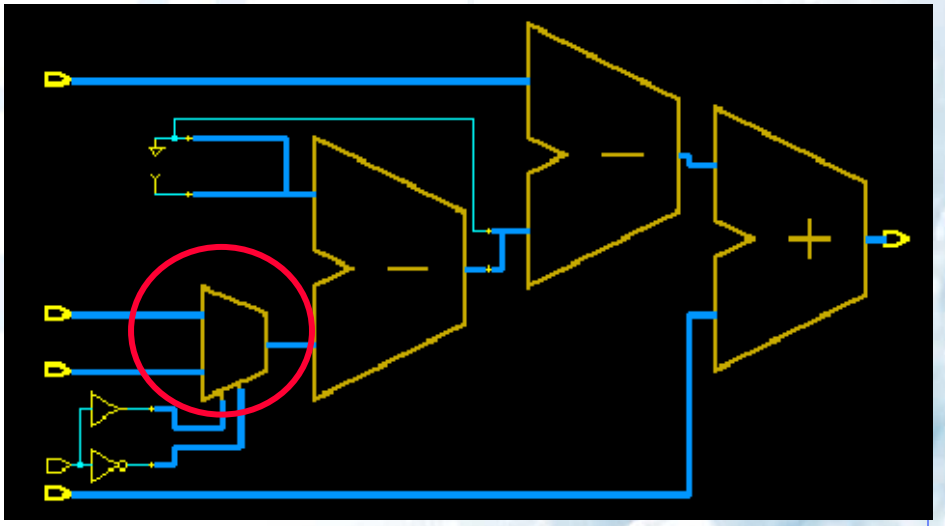
Coding Skill-Data-path Duplication(2/2)

- We assume that signal "CONTROL" is the latest arrival pin.
- By this skill, we will reduce latency but we must pay for it, area!

Dulpicated



No_dulpicated



Coding Skill -- operator in `if` (1/2)

- We assume that signal "A" is latest arrival signal

Before_improved

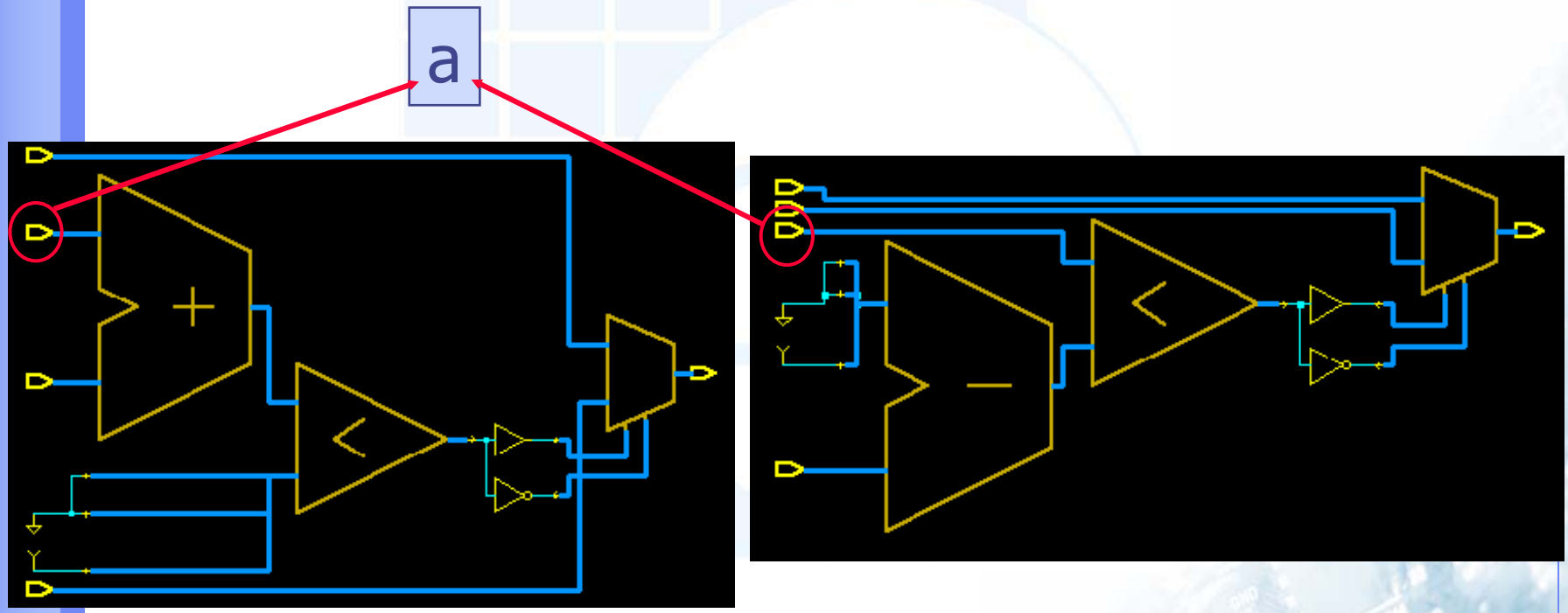
```
module cond_oper(A, B, C, D, Z);
parameter N = 8;
input [N-1:0] A, B, C, D;
//A is late arriving
output [N-1:0] Z;
reg [N-1:0] Z;

always @(A or B or C or D) begin
if (A + B < 24)
  Z <= C;
else
  Z <= D;
end
endmodule
```

Improved

```
module cond_oper_improved (A, B, C, D, Z);
parameter N = 8;
input [N-1:0] A, B, C, D;
// A is late arriving
output [N-1:0] Z;
reg [N-1:0] Z;

always @(A or B or C or D) begin
if (A < 24 - B)
  Z <= C;
else
  Z <= D;
end
endmodule
```

Coding Skill -- operator in `if` (2/2)

- In this example, not only latency reduced, but also area reduced.



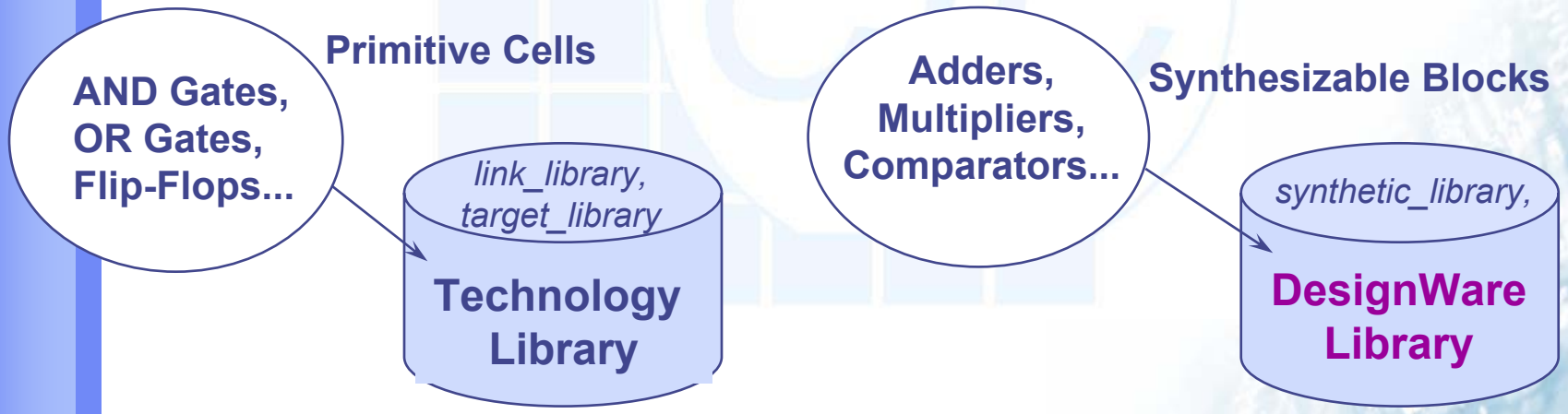
Before_improved

Improved

DesignWare Library

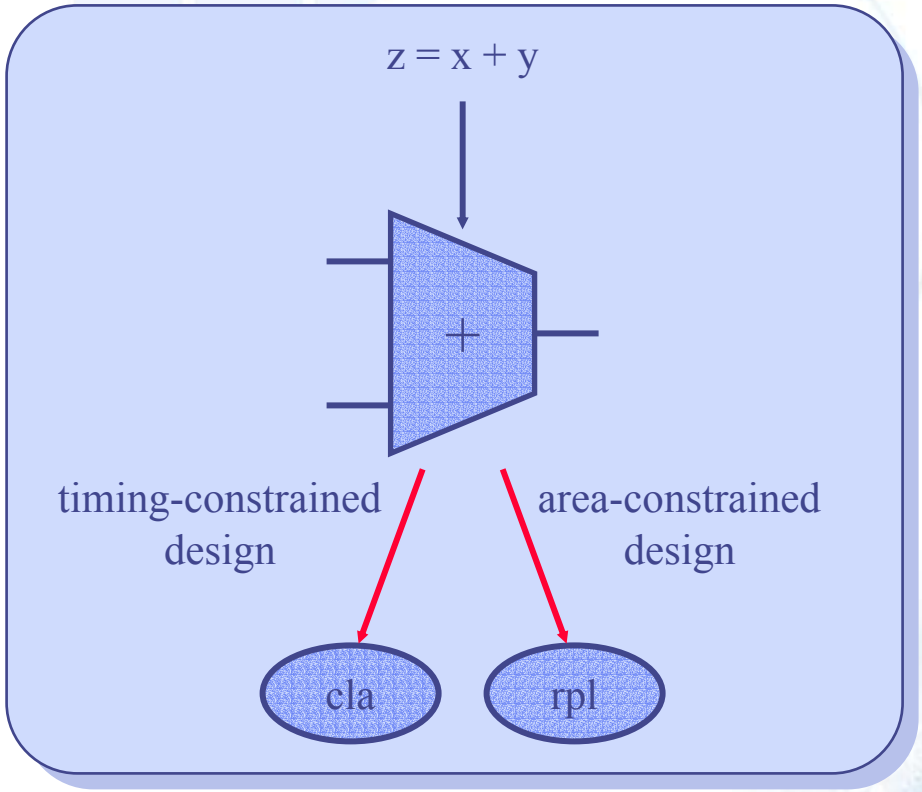
DesignWare Library (1/3)

- DesignWare is technology-independent "soft macros" such as adders, comparators, etc., which can be synthesized into gates from your target library.
- Enable the user to imply large and complex arithmetic operations to be synthesized



DesignWare Library (2/3)

- Multiple architectures for each macro allow DC to evaluate speed/area tradeoffs and choose the best implementation



DesignWare Library (3/3)

- If you want to use the DesignWare library, you must set the "synthetic_library" and "search_path" in the .synopsys_dc.setup
- Example

```
synthetic_library = {"dw_foundation.sldb"}
```

- If two modules in different libraries have the same name, the module in the first library listed is used.

DesignWare Part (1/4)

- The name of DesignWare part is based upon
  - The name of the design module
  - The type of the design synthesized
  - A unique decimal extension
- Example:

```
module adder(z,a,b,c);
input [7:0] a,b,c;
output [7:0] z;

assign z=a+b+c;

endmodule
```

DesignWare Part (2/4)

- Invoke DesignWare component with two ways
  - Inference: let design compiler to choose the DesignWare component according to the constraints
  - Instantiation: explicitly instantiate synthetic modules

DesignWare Part (3/4)

- Example

Inference

```
module adder(in1,in2,sum);
parameter wordlength = 8;
  input [wordlength-1:0] in1,in2
  output [wordlength-1:0] sum;
  assign sum = in1 +in2;
endmodule
```

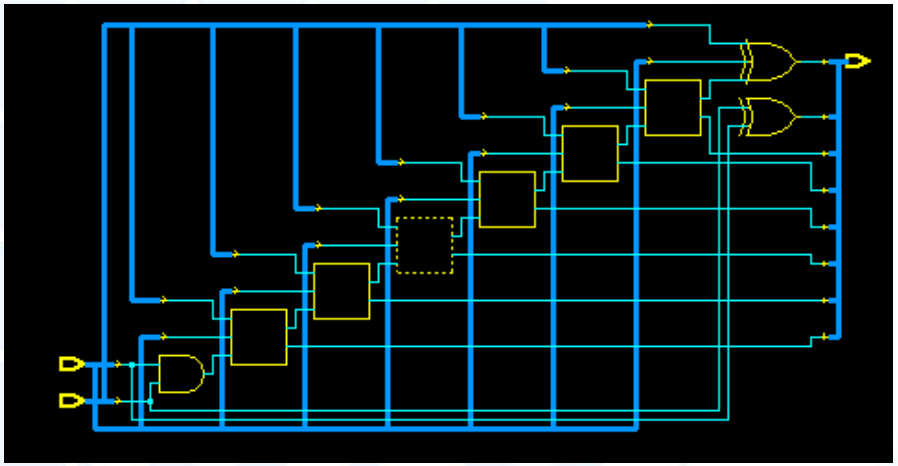
Instantiation

```
module adder(in1, in2, carry_in, sum, carry_out);
  parameter wordlength = 8;
  input [wordlength-1:0] in1, in2;
  input carry_in;
  output [wordlength-1:0] sum;
  output carry_out;
  DW01_add #(wordlength) U1(in1,in2,carry_in,sum,carry_out);
endmodule
```

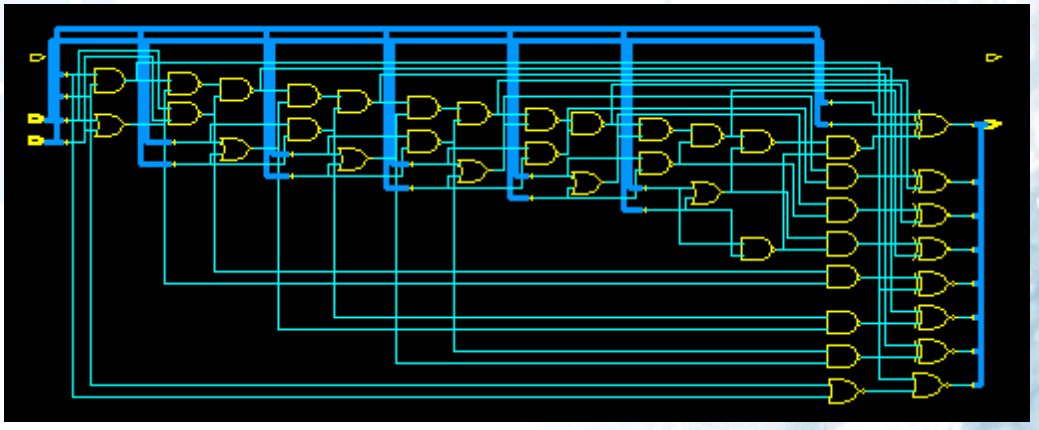
DesignWare Part (4/4)

- Example : assign c = a + b;

Optimized for area only 8-bit ripple adder



Optimized for speed 8-bit CLA adder



DesignWare Report

- Analysis/Report/Resource -- report_resource

Report Output

```
design_analyzer>
****************************************
Report : resources
Design : lab1
Version: 2002.05-SP1
Date   : Wed May 14 17:24:53 2003
****************************************

Resource Sharing Report for design lab1 in file
        /users2/cic/ccyang/SYNOPSYS/LAB/lab1/lab1.v
===============================================================================
|          |              |            | Contained     |                      |
| Resource | Module       | Parameters | Resources     | Contained Operations |
===============================================================================
| r236     | DW01_add     | width=9    |               | add_8 add_9          |
===============================================================================

Implementation Report
===============================================================================
|                  |                  | Current          | Set              |
| Cell             | Module           | Implementation   | Implementation   |
===============================================================================
| r236             | DW01_add         | rpl              |                  |
===============================================================================
```

Show | Next | Previous | Cancel

DW-Select implementation (1/2)

- In using DesignWare Library, we can select our favorite implementation for component, such as when we use a component "dw01_add", we also can further specify that the adder is a cla-adder or a rpl-adder.
- How ?
  - Embedded in your RTL code
  - Use "set_implementation" dc_shell command
- All information about DesignWare is included in the Synopsys On-Line Documentation (SOLD) -- "DesignWare" part

DW-Select implementation (2/2)

Table 4 - Synthesis Implementations

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Basic
cla	Carry look-ahead synthesis model	DesignWare-Basic
clf	Fast carry look-ahead synthesis model	DesignWare-Foundation
bk	Brent-Kung architecture synthesis model	DesignWare-Foundation
csm[a]	Conditional sum synthesis model	DesignWare-Foundation
rpcs	Ripple carry select architecture	DesignWare-Foundation
clsa[b]	MC inside DW Carry-Lookahead-Select	DesignWare-Foundation
csa[b]	MC inside DW Carry-Select	DesignWare-Foundation
fastcla[b]	MC inside DW Fast Carry-Lookahead	DesignWare-Foundation

Table 5 - Simulation Models

Model	Function
DW01.DW01_ADD_CFG_SIM	Design unit name for VHDL simulation
dw/dw01/src/DW01_add_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW01_add.v	Verilog simulation model source code

Implementation – embedded (1/2)

- Choose a implementation you want
- Learn coding style -- a resource can be declared only in an "always" block
  - Specify the resource name

    /* synopsys resource resource_name */
  - select component

    /* map_to_module = "module_name" */
  - select implementation_name

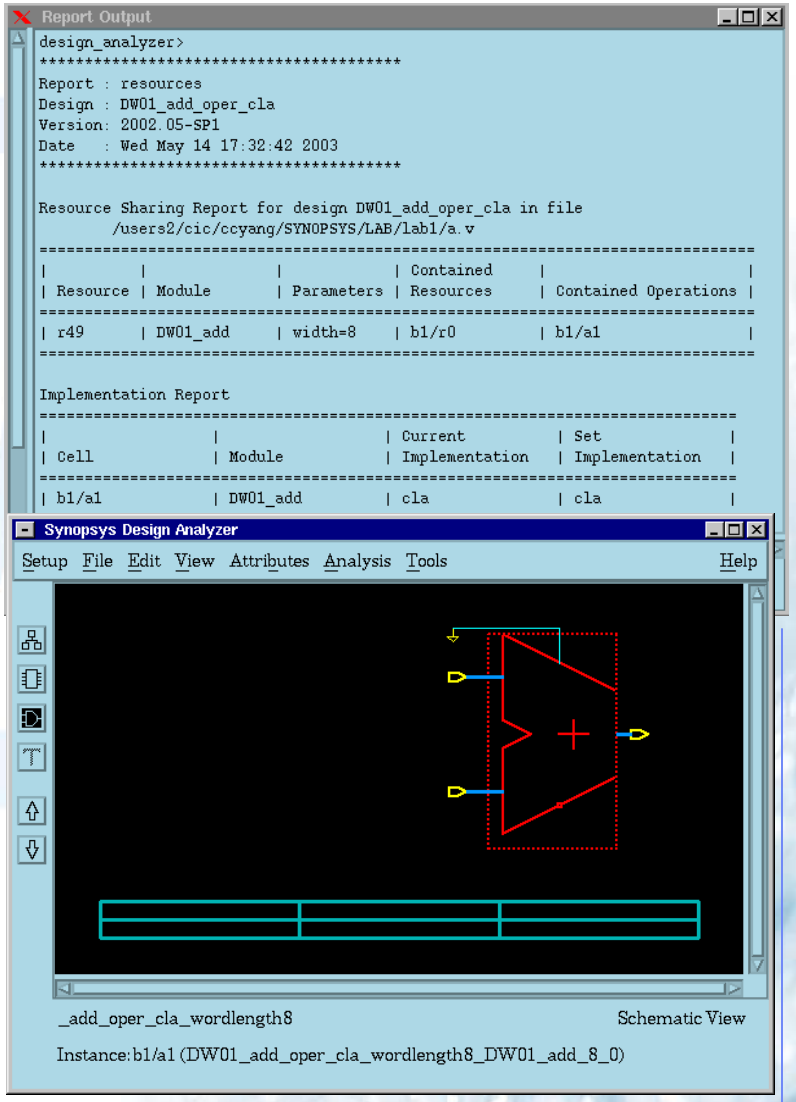
    /* implementation = "impl_name" */
  - bind labeled operation to the specific module & implementation /* ops = "label_name" ;*/
  - label the operation

    z = a + b; // synopsys label label_name

Implementation – embedded (2/2)

- Example -- an adder with component “dw01_add” and with implementation “ cla “

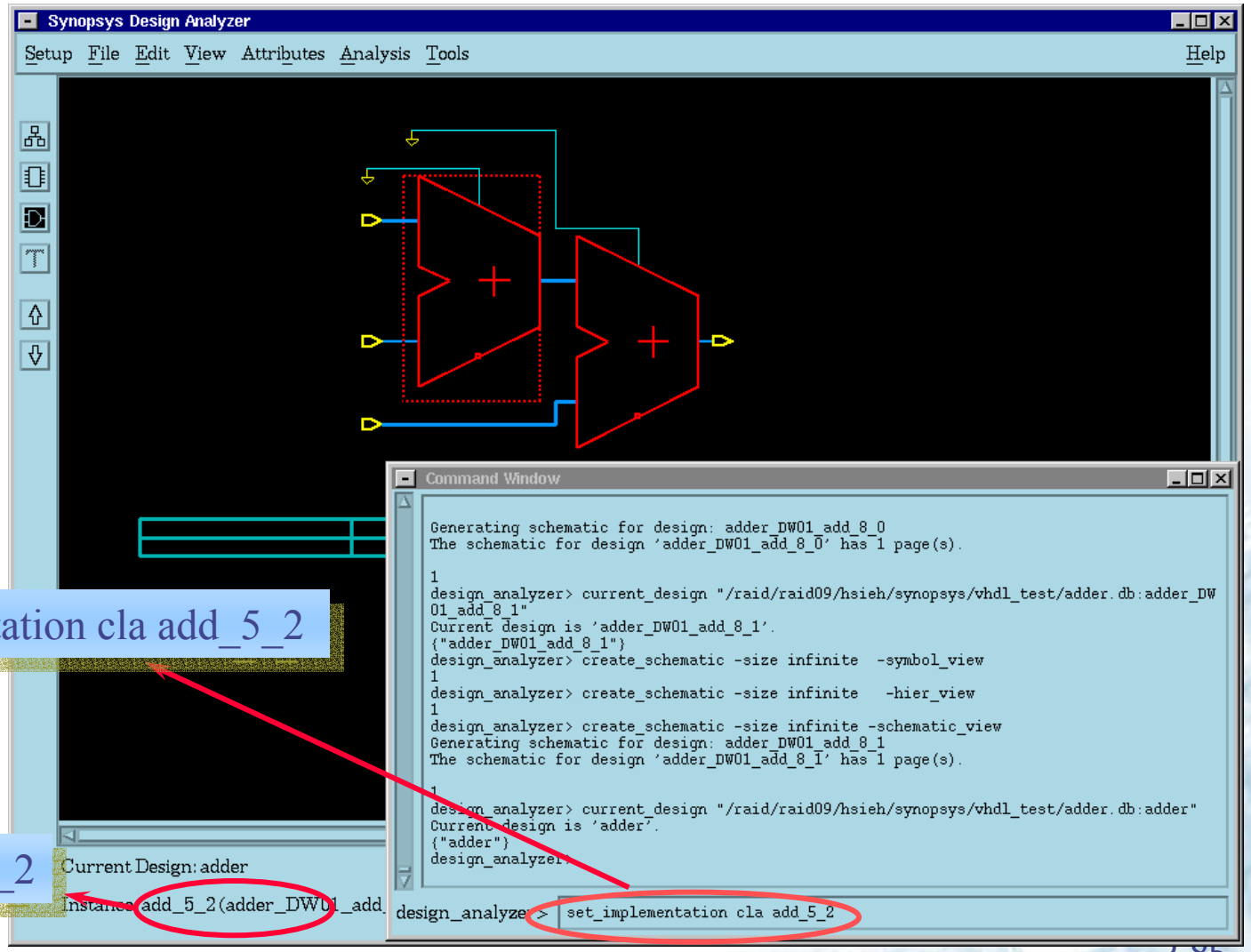
```
module DW01_add_oper_cla(in1,in2,sum);
  parameter wordlength = 8;
  input [wordlength-1:0] in1,in2;
  output [wordlength-1:0] sum;
  reg [wordlength-1:0] sum;
   always @(in1 or in2) begin :b1
   /* synopsys resource r0:
      map_to_module = "DW01_add",
      implementation = "cla",
      ops = "a1"; */
      sum <= in1 + in2; //synopsys label a1
   end
endmodule
```



DesignWare – set_implementation (1/2)

- Specify the implementation format of your DesignWare component
  - Choose the cell you want to specify the implementation style and then see what its instance_name is
  - Choose the implementation style by the Synopsys On-Line-Documentation (SOLD)– DesignWare part
  - Use a dc_shell command to specify the implementation style
    - " *set_implementation implementation_name instance_name* "
  - compile
  - report -- resource

DesignWare – set_implementation (2/2)



set_implementation cla add_5_2

Instance:add_5_2

DesignWare Simulation

- Inference : just as usual, do not need any special handling
- Instantiation : in your $SYNOPSYS directory, you can find the designware's simulation model, and then include this simulation model to do your simulation.
  - The exact directory is
    - $SYNOPSYS/dw/dw0x/src -- for VHDL
    - $SYNOPSYS/dw/dw0x/src_ver -- for Verilog

```
//synopsys translate_off
`include "/usr/synopsys/synthesis/cur/dw/dw01/src_ver/DW01_sub.v"
`include "/usr/synopsys/synthesis/cur/dw/dw02/src_ver/DW02_mult.v"
`include "/usr/synopsys/synthesis/cur/dw/dw02/src_ver/DW_div.v"
`include "/usr/synopsys/synthesis/cur/dw/dw02/src_ver/DW02_mac.v"
`include "/usr/synopsys/synthesis/cur/dw/dw02/src_ver/DW_square.v"
//synopsys translate_on
```

Design Constraints Settings

Design Constraints Setting

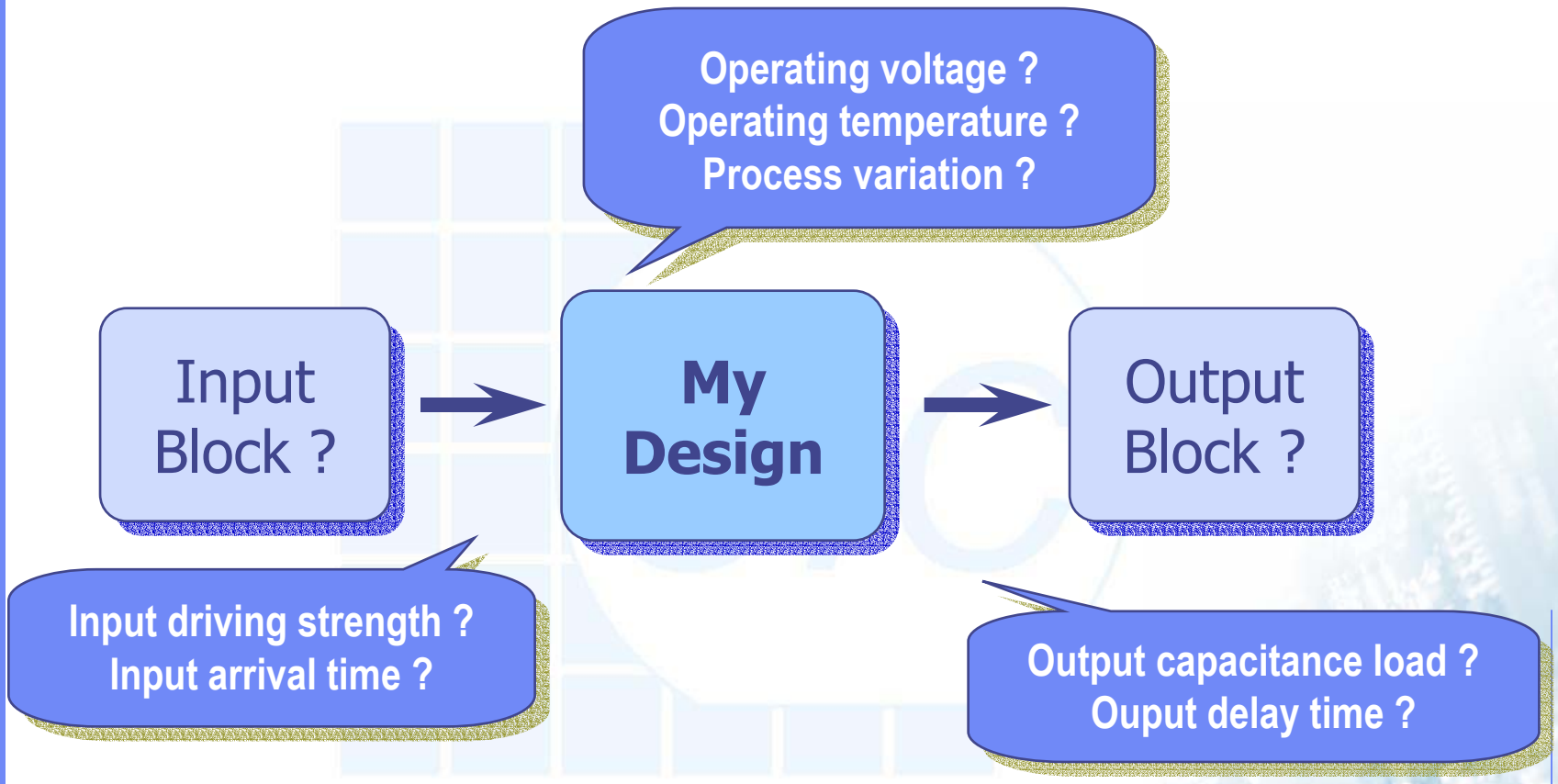
- ➔Setting Design Environment
- ➔Setting Design Constraint

Setting Design Environment

Why Describes the Real World Environment

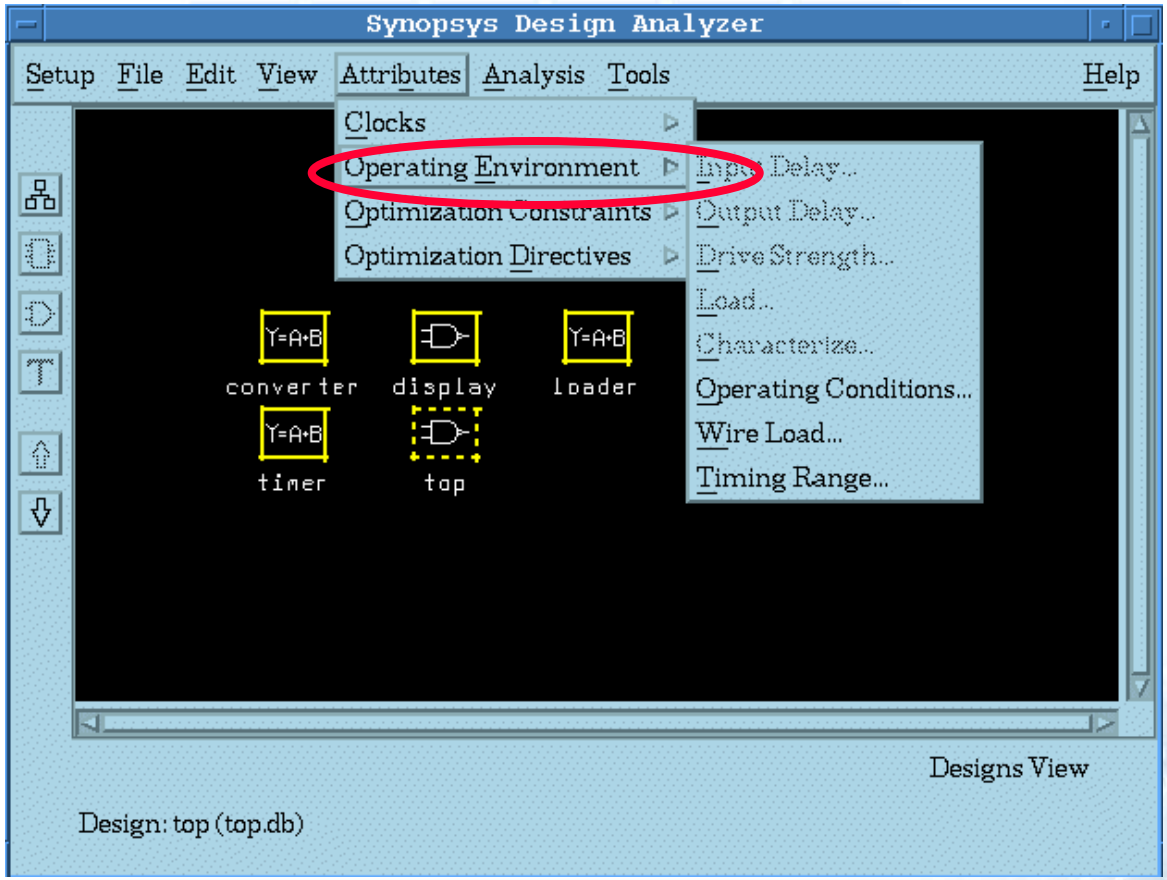
- Beware that the defaults are not realistic conditions.
  - Input drive is not infinite
  - Capacitive loading is usually not zero
  - Consider process, temperature, and voltage variation
- The operating environment affects the components selected from target library and timing through your design.
- The real world environment you define describes the conditions that the circuit will operate within.

Describing Design Environment



Setting Operating Environment

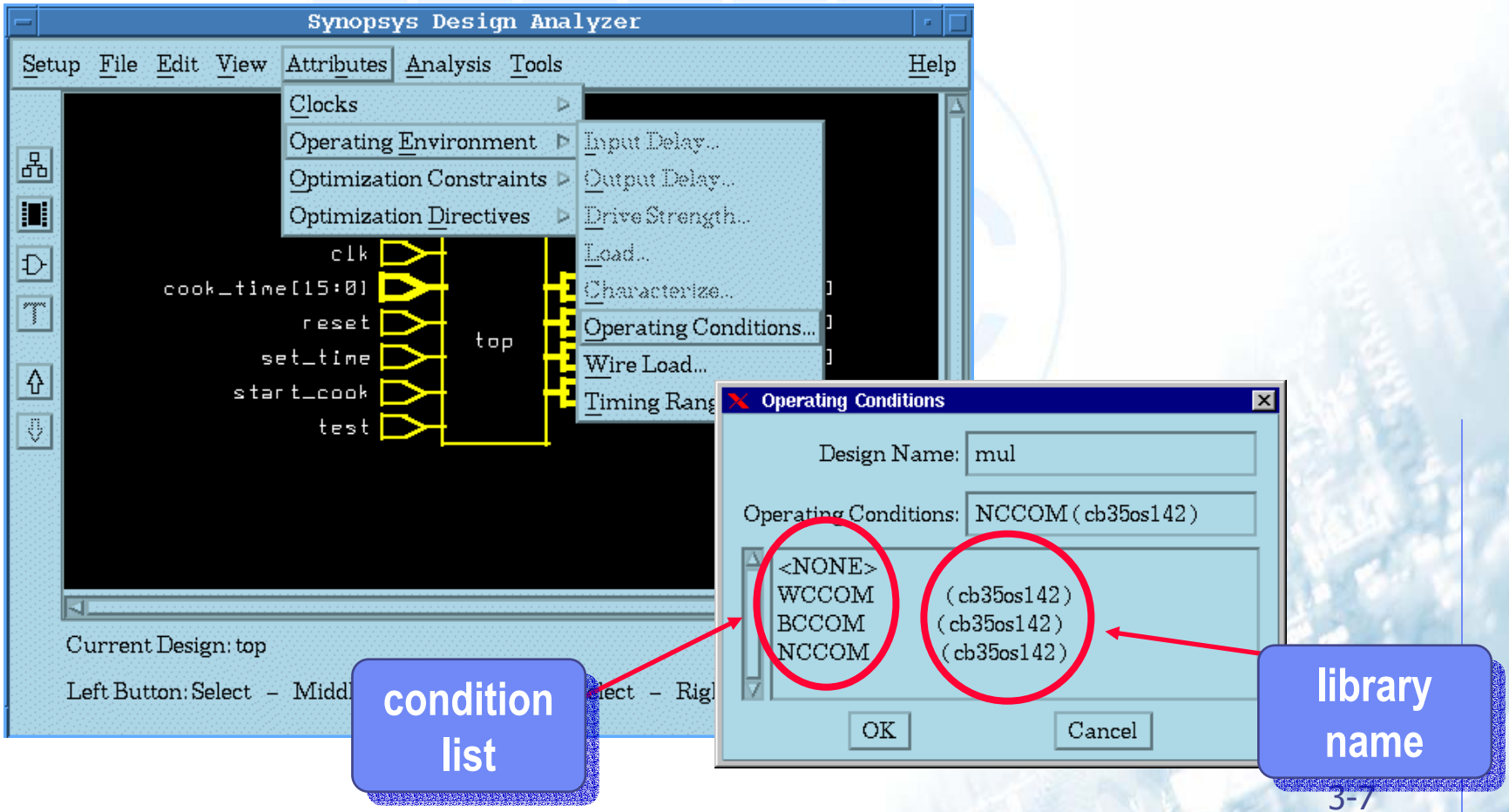
- Attributes/Operating Environment



Setting Operating Condition

- Attributes/Operating Environment/Operating Condition

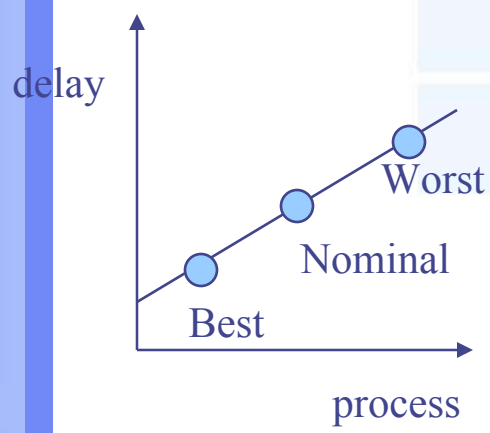
```
dc_shell> set_operating_conditions "WCCOM"
```

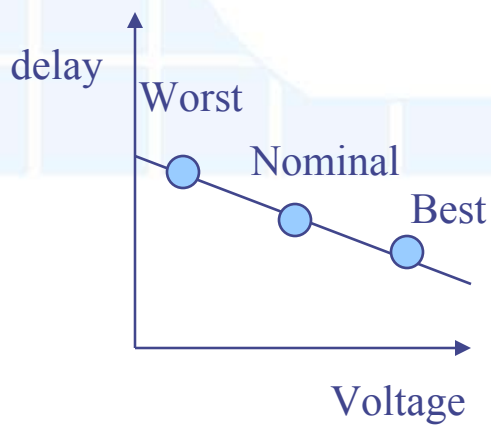


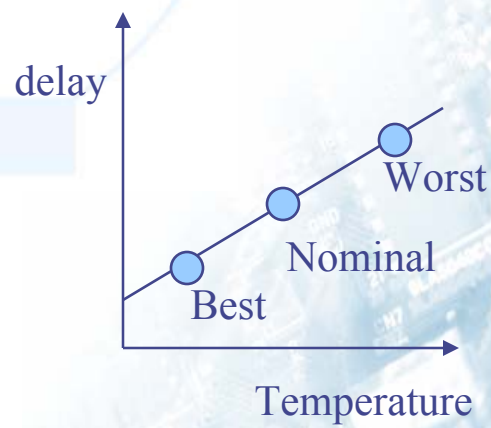
Operating Condition

- Operating condition model scales components delay, directs the optimizer to simulate variations in process, temperature, and voltage.

Name	*Process*	*Temp*	*Volt*	*Interconnection Model*
WCCOM	**1.32**	**100.00**	**2.7**	**worst_case_tree**
BCCOM	**0.73**	**0.00**	**3.6**	**best_case_tree**
NCCOM	**1.00**	**25.00**	**3.3**	**balance_tree**

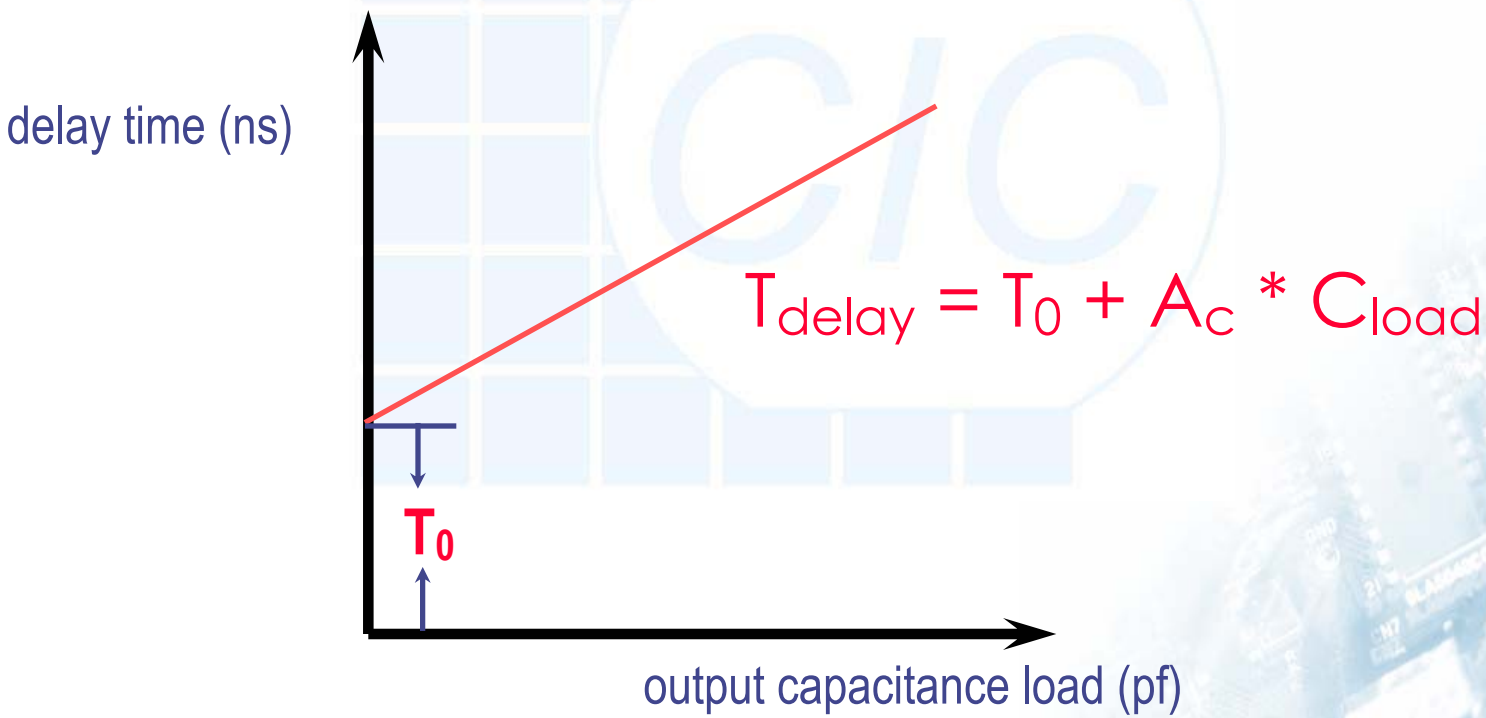






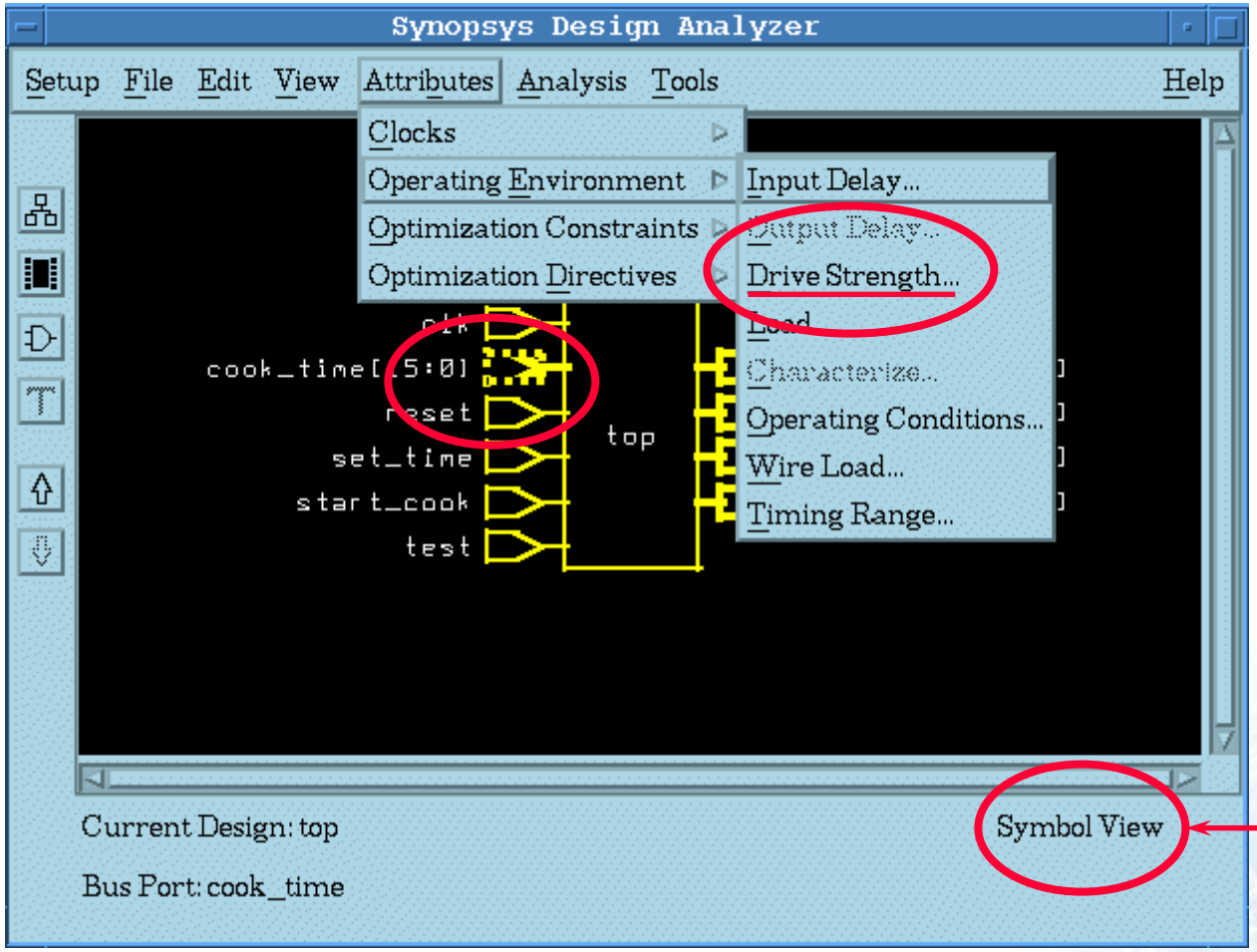
Input Drive Impedance

- $T_{delay} = T_0 + A_c * C_{load}$
  - T_0 : cell pin to pin intrinsic delay
  - A_c : drive impedance (unit: ns/pf)



Setting Input Drive Impedance (1/2)

- Attribute/Operating Environment/Drive Strength



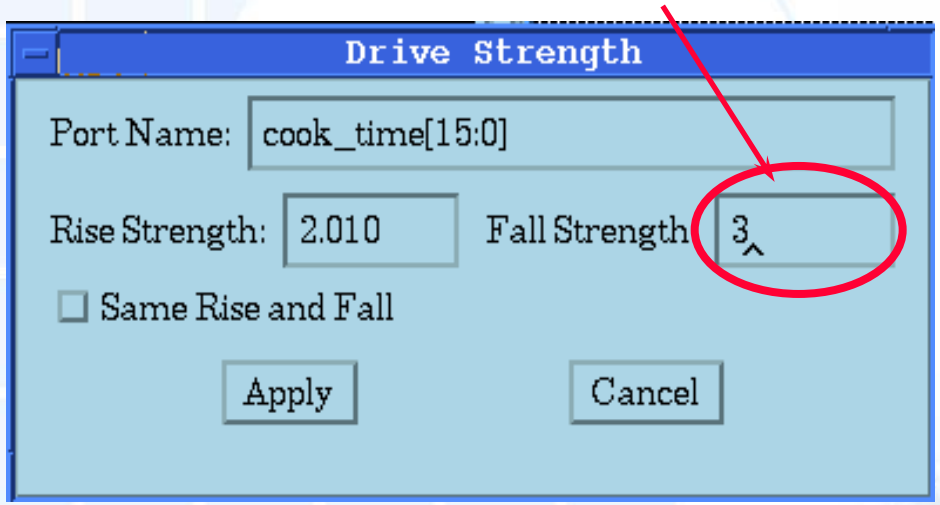
Set attributes in symbol view

Setting Input Drive Impedance (2/2)

- Also can be set using "drive_of" command
- Example: (buffd1 cell output)

```
drive_of (cb35os142_typ/buffd1/Z)
```

(unit: **ns/pf**)

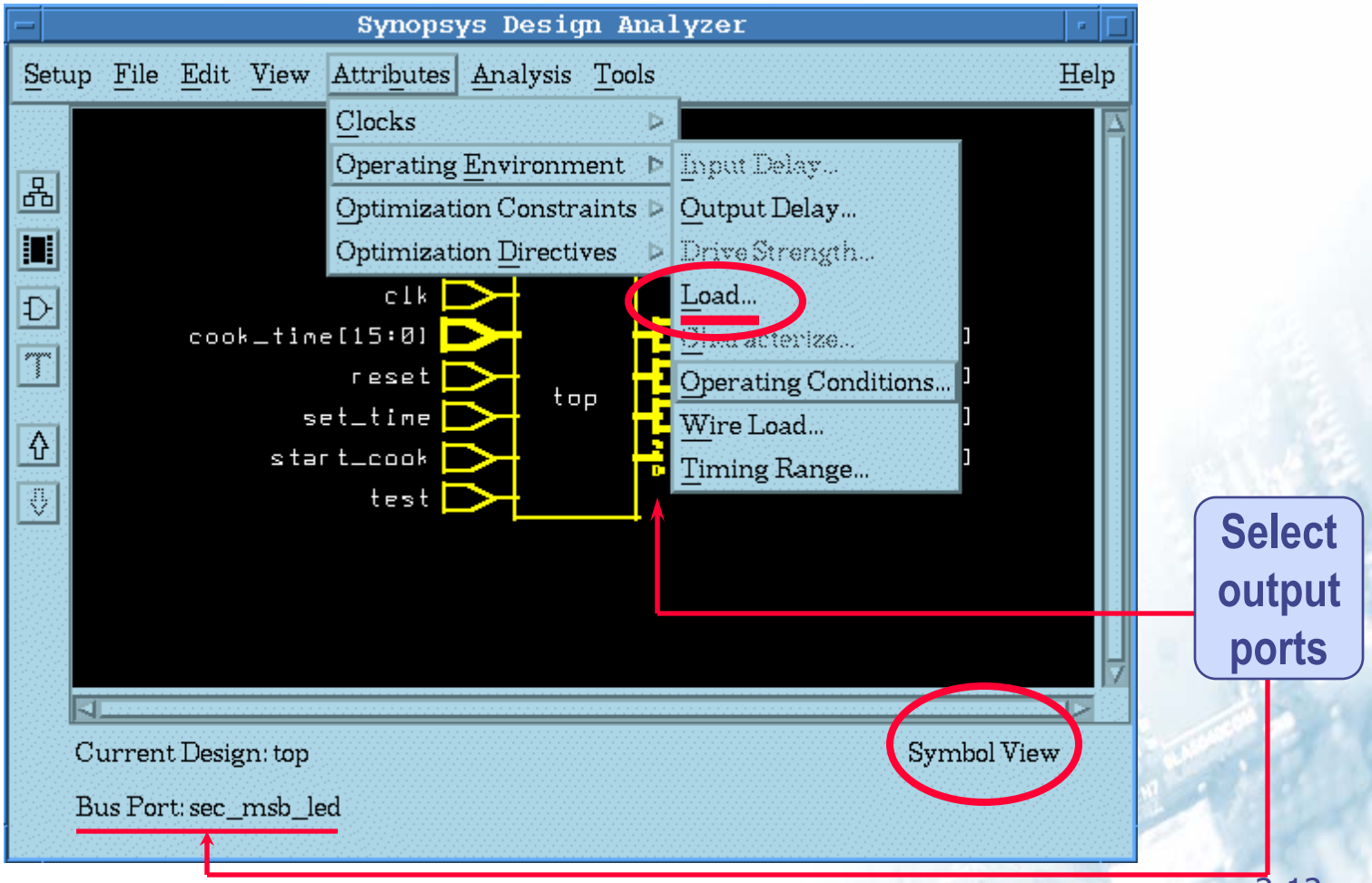


(Fill in the blank and "ENTER")

```
dc_shell> set_drive_cell -lib cb35os142 -cell buffd1 -pin Z  \
                         find (port, cook_time[15:0])
```

Setting Output Loading (1/2)

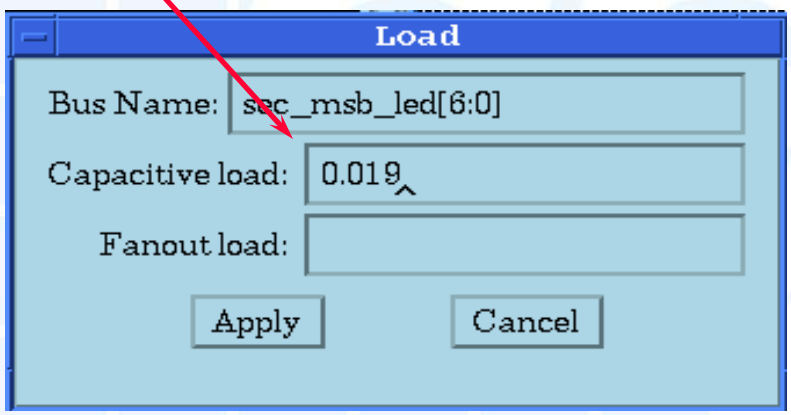
- Attribute/Operating Environment/Load



Setting Output Loading (2/2)

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

```
load_of (cb35os142_typ/buffd1/I)
```



(unit : *pf*)

(Fill in the blank and "ENTER")

```
dc_shell> set_load load_of (cb35os142/buffd1/I) \
                   find(port, set_msb_led[6:0])
```

Port Report

- dc_shell command

```
dc_shell> report_port -verbose { port_list }
```

or in the option menu set *verbose*

Command Window

```
Performing report_port on port 'sec_lsb_led[0]'.

****************************************
Report : port
        -verbose
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 17:57:19 2003
****************************************


                                Pin      Wire     Max      Max      Connection
Port            Dir             Load     Load     Trans    Cap      Class       Attrs
-----------------------------------------------------------------------------------------
min_msb_led[6]  out             0.0110   0.0000   --       --       --
min_msb_led[5]  out             0.0110   0.0000   --       --       --
min_msb_led[4]  out             0.0110   0.0000   --       --       --
min_msb_led[3]  out             0.0110   0.0000   --       --       --
min_msb_led[2]  out             0.0110   0.0000   --       --       --
min_msb_led[1]  out             0.0110   0.0000   --       --       --
min_msb_led[0]  out             0.0110   0.0000   --       --       --
min_lsb_led[6]  out             0.0110   0.0000   --       --       --
min_lsb_led[5]  out             0.0110   0.0000   --       --       --
min_lsb_led[4]  out             0.0110   0.0000   --       --       --
min_lsb_led[3]  out             0.0110   0.0000   --       --       --
min_lsb_led[2]  out             0.0110   0.0000   --       --       --
min_lsb_led[1]  out             0.0110   0.0000   --       --       --
min_lsb_led[0]  out             0.0110   0.0000   --       --       --
sec_msb_led[6]  out             0.0110   0.0000   --       --       --
sec_msb_led[5]  out             0.0110   0.0000   --       --       --
sec_msb_led[4]  out             0.0110   0.0000   --       --       --
sec_msb_led[3]  out             0.0110   0.0000   --       --       --
sec_msb_led[2]  out             0.0110   0.0000   --       --       --
design_analyzer>
```

Command Window

```
                 Max Drive        Min Drive        Resistance       Min      Min      Cell
Input Port       Rise    Fall     Rise    Fall     Max     Min      Cap      Fanout   Deg
-------------------------------------------------------------------------------------------
clk              0.47    0.47     --      --       --      --       --       --       --
cook_time[0]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[1]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[2]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[3]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[4]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[5]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[6]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[7]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[8]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[9]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[10]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[11]
                 3.54    3.54     --      --       --      --       --       --       --
design_analyzer>
```

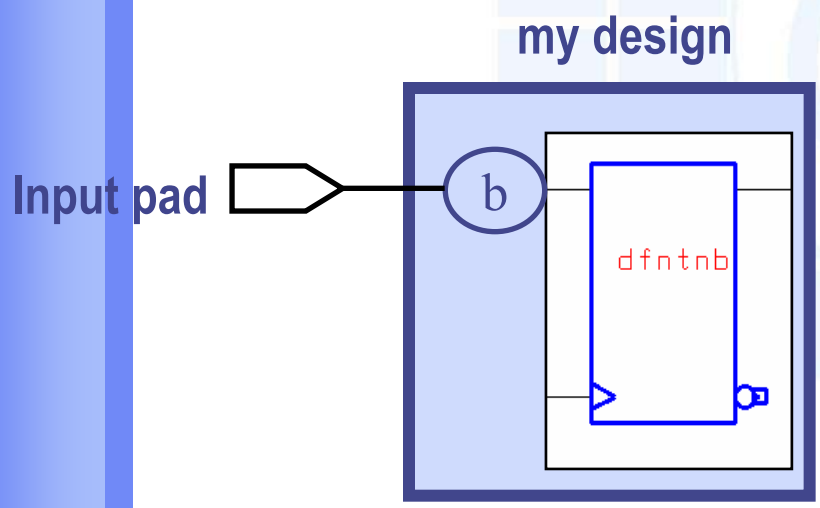
Drive Strength & Load for Pads

- How do you specify the Drive Strength & Output Load for the pins which connect to pads ?
- Example:
  - In CIC's cell_based design flow, we use the Avant! 0.35um cell library. In this library we can find a file named "*ds_cb35io122.pdf*", *and* in this file we can find the information for the pads you want to use.

  1. Based on the information, set the input drive strength & output load.
  2. Or extract the pad boundary condition by using ***characterize*** command, we'll introduce it later.

Input Drive Strength for Pads

- If your design is as follows:
- Assumed that we use input pad PC3D01, by the list as follow, we can set the input drive strength as 0.2468 (ns/pf)



3V CMOS Input Only Pads

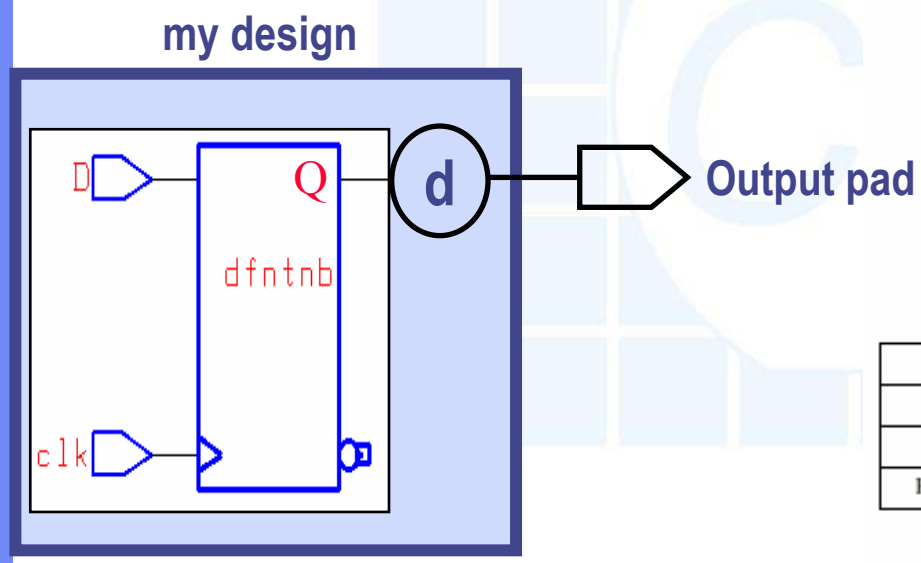
PC3D01, PC3D11, PC3D21, and PC3D31

Performance Equations

PC3D01		RISE	FALL
tCD	PAD -> CIN	0.3377 + 0.0378 + 0.2468 * Cld	0.2596 + 0.0303 + 0.1978 * Cld
PC3D11		RISE	FALL
tCD	PAD -> CIN	0.3261 + 0.0292 + 0.2265 * Cld	0.3782 + 0.0253 + 0.1960 * Cld
PC3D21		RISE	FALL
tCD	PAD -> CIN	0.7915 + 0.0374 + 0.2618 * Cld	0.6064 + 0.0395 + 0.2763 * Cld
PC3D31		RISE	FALL
tCD	PAD -> CIN	0.6828 + 0.0289 + 0.2256 * Cld	0.7971 + 0.0253 + 0.1976 * Cld

Output Load for Pads

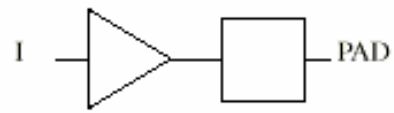
- If your design is as follow:
- Assumed that we use output pad PC3O01, by the list as follow, we can set the output load as 0.096



3V CMOS Output Pads

PC3O01 through PC3O05

PC3O01 through PC3O05 cells are CMOS output pads with AC drive capbilities ranging from 1x to 5x.



Function Table

INPUT	OUTPUT
I	PAD
L	L
H	H

Cell Description

Macro Name:	PC3O01	PC3O02	PC3O03	PC3O04	PC3O05
Drive Capability	1x	2x	3x	4x	5x
Width (mils):	3.4	3.4	3.4	3.4	3.4
Power (µW/MHz):	198.55	198.59	198.75	198.81	197.58

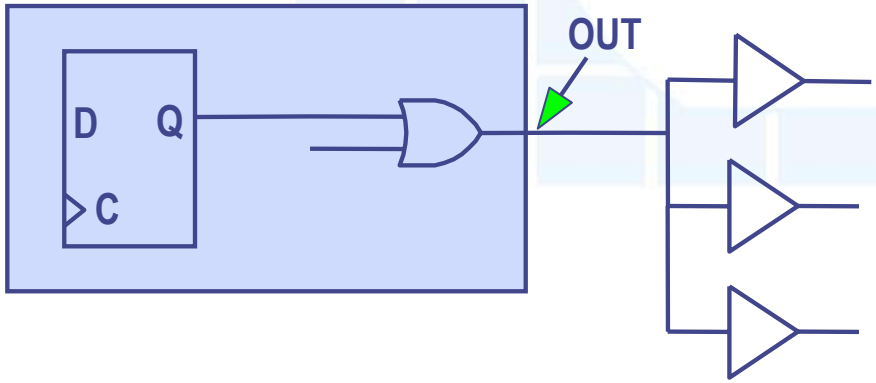
Pin Description

name	Capacitance (pF)					Description
	PC3O01	PC3O02	PC3O03	PC3O04	PC3O05	
I	0.096	0.096	0.096	0.132	0.133	Input
PAD	8.577	8.577	8.577	8.576	8.573	Output

Setting the Fanout Load

- Use fanout_load to regulate max_fanout.
- Syntax: `set_fanout_load fanout portlist`
  (for listed output ports)
- Design Rule: external fanout_load + internal fanout_load must be smaller than max_fanout_load
  - "OUT" has external fanout_load 4.5 (1.5 x 3)
  - "OUT" has external capacitance 6 (2 x 3)
  - "OUT" has the number of fanout 3 (3 buffers)

Different!

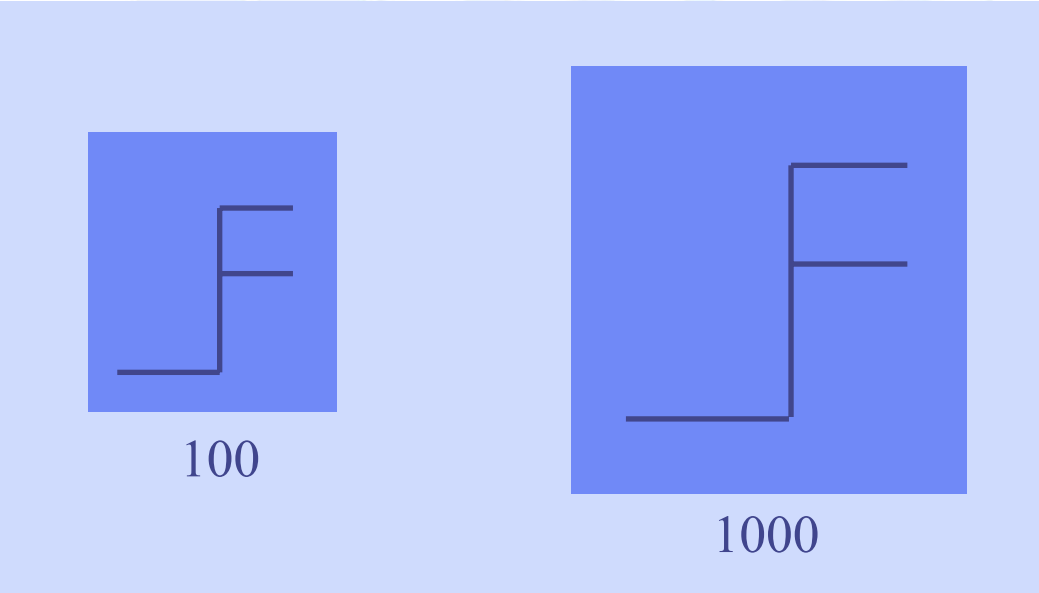


Each buffer has
`fanout_load = 1.5`
`capacitance =2`
for the input

```
dc_shell> set_fanout_load 4.5 find(port, out)
```

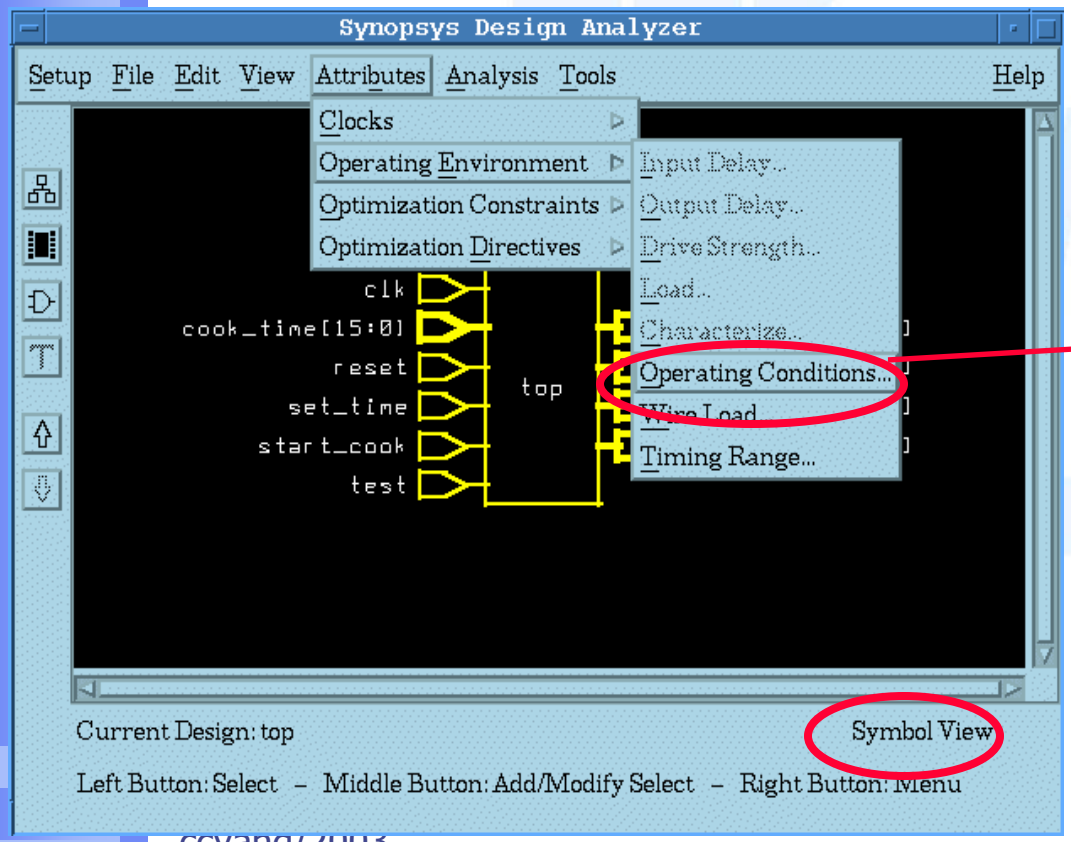
Wire Load Model

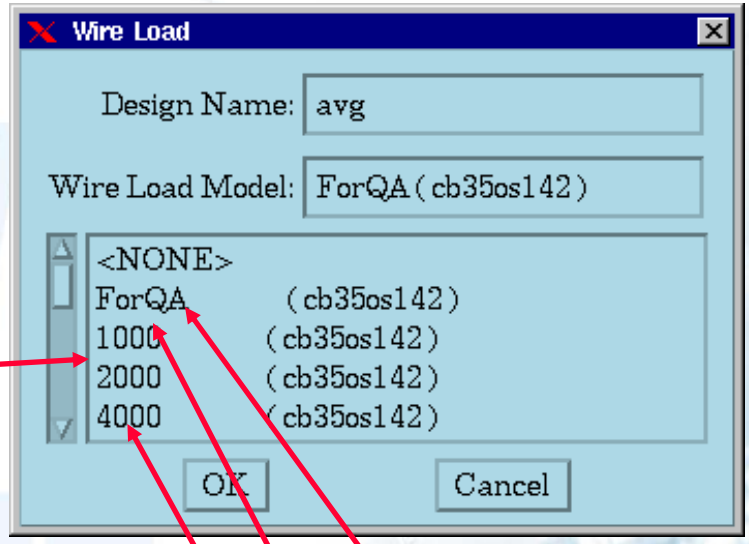
- Wire load model estimates wire capacitance based on chip area & cell fanout.
- Setting this information during compile in order to model the design more accurately .



Setting Wire Load Model (1/2)

- Attributes/Operating Environment/Wire Load
- If you don't specify the wireload model, Design Compiler will select wire load model automatically according to your compiled chip area.



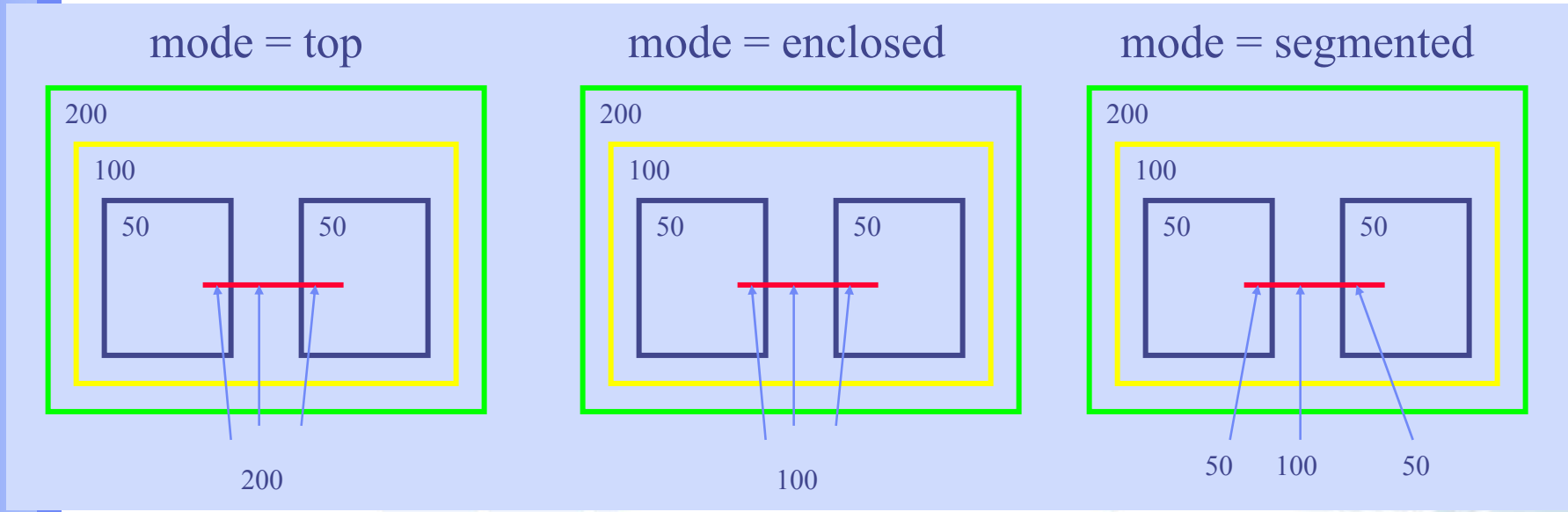


Design fewer than 1000 gates

1000~2000 gates design

4000~8000 gates design

Setting Wire Load Model (2/2)



- Use **-mode** option, you can specify which wire load mode to use for nets that cross hierarchical boundaries

```
dc_shell> set_wire_load  ForQA  -mode top
```

Wire Load Model Example (Design Time)

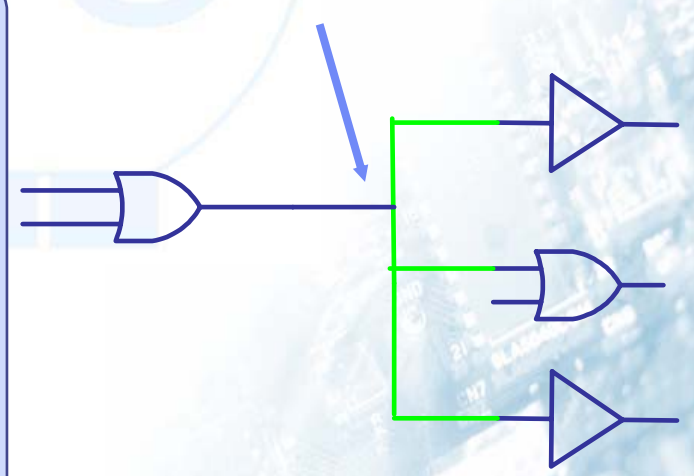
- To calculate the R, C and net area of a net
  1. Determine the number of fanout of the net
  2. Look up the length from the fanout_length pairs in the wire_load model
  3. Multiply the length by the capacitance (or R or area)coefficient

Cwire = (fanout=3 ➔ length =2.8) x capacitance coefficent (1.3) =3.64 load units

Rwire = (fanout=3 ➔ length =2.8) x resistance coefficient (3.0) = 8.4 resistance units

Net area = (fanout=3 ➔ length =2.8) x area coefficient (0.04) = 0.112 net area units

```
wire_load ("500") (
resistance : 3.0   /* R perunit length*/
capacitance: 1.3   /* C per unit length */
area: 0.04         /*area per unit length */
slop: 0.15         /* extrapolatoin slope*/
fanout_length ( 1 ,  2.1 )  /* fanout-length
pairs */
fanout_length ( 2 ,   2.5)
fanout_length ( 3 ,   2.8)
fanout_length ( 4 ,   3.3)
```



Setting Design Constraint

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.

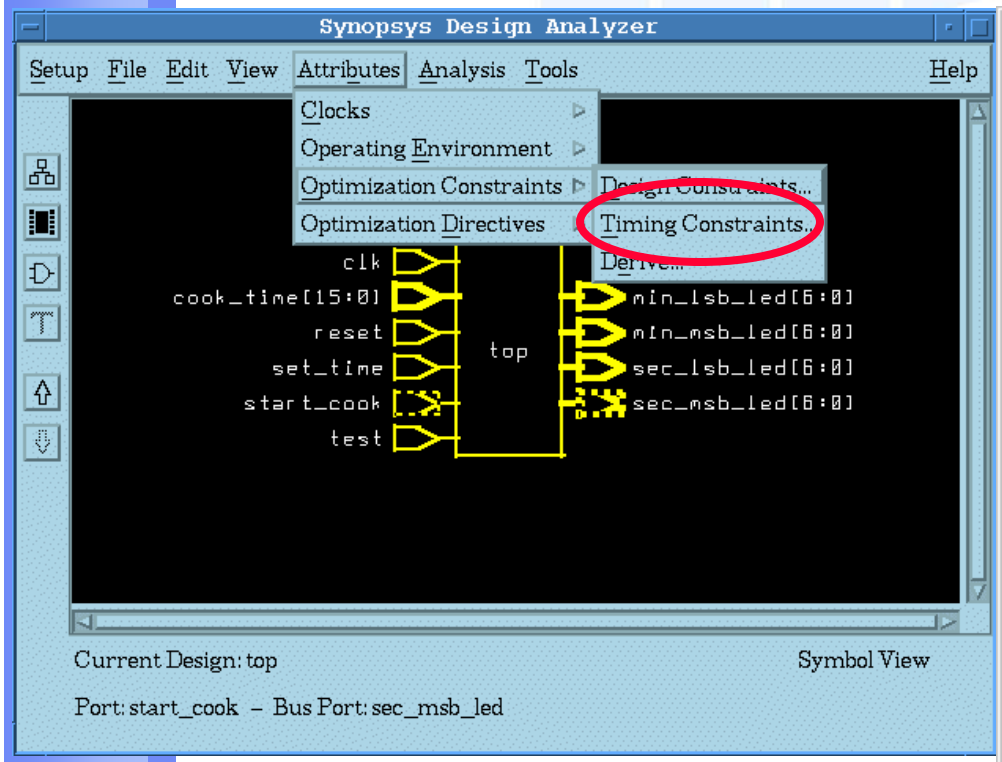
Optimization Constraints

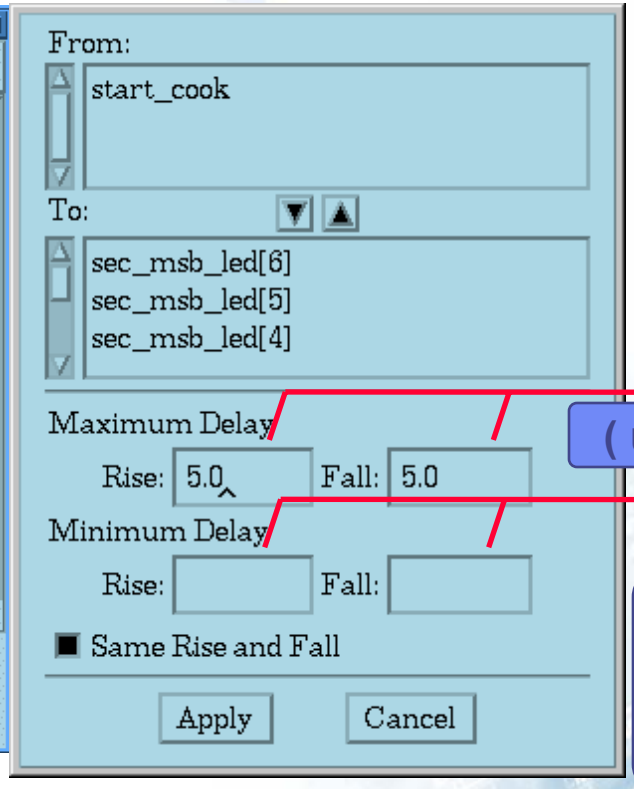
- Optimization constraints, in order of attentions are
  1. Maximum delay
  2. Minimum delay
  3. (Maximum power)
  4. Maximum area

- About combinational circuit, we only set maximum delay & minimum delay for timing constraint

Maximum Delay Constraints

- For combinational circuits primarily
  - Select the start & end points of the timing path
  - Attributes/Optimization Constraints/Timing Constraints





maximum delay constraint

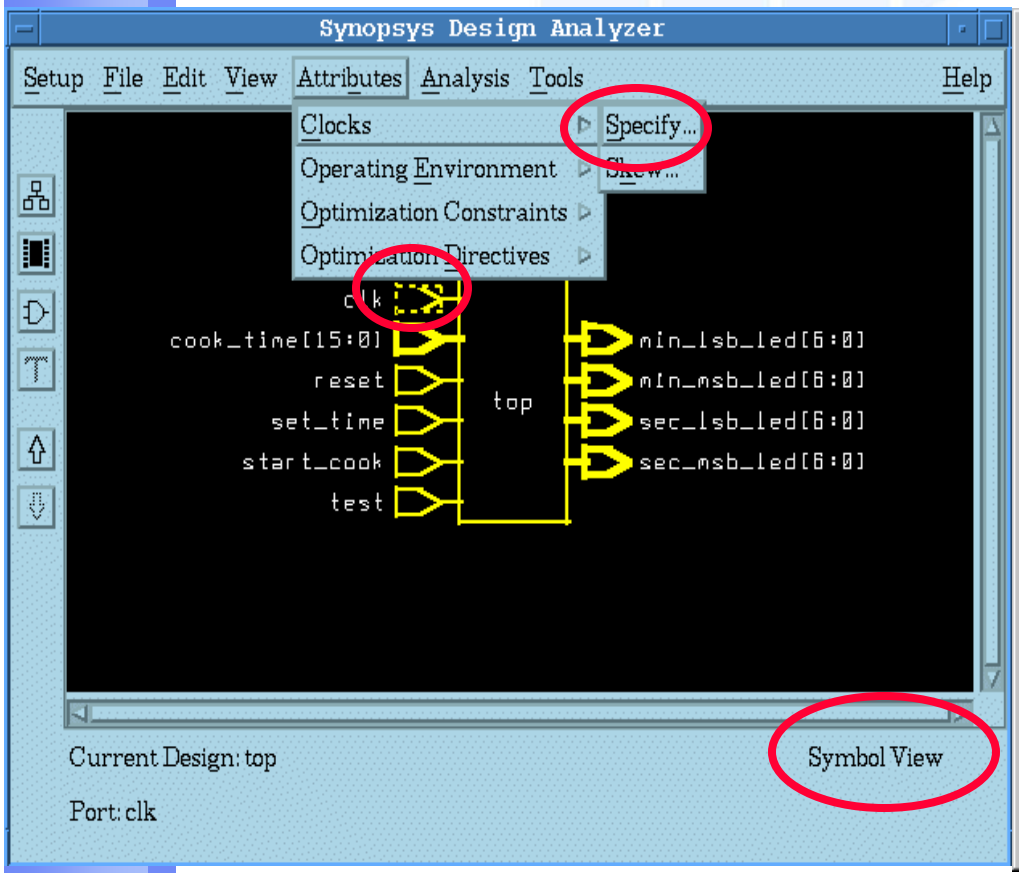
(unit : *ns*)

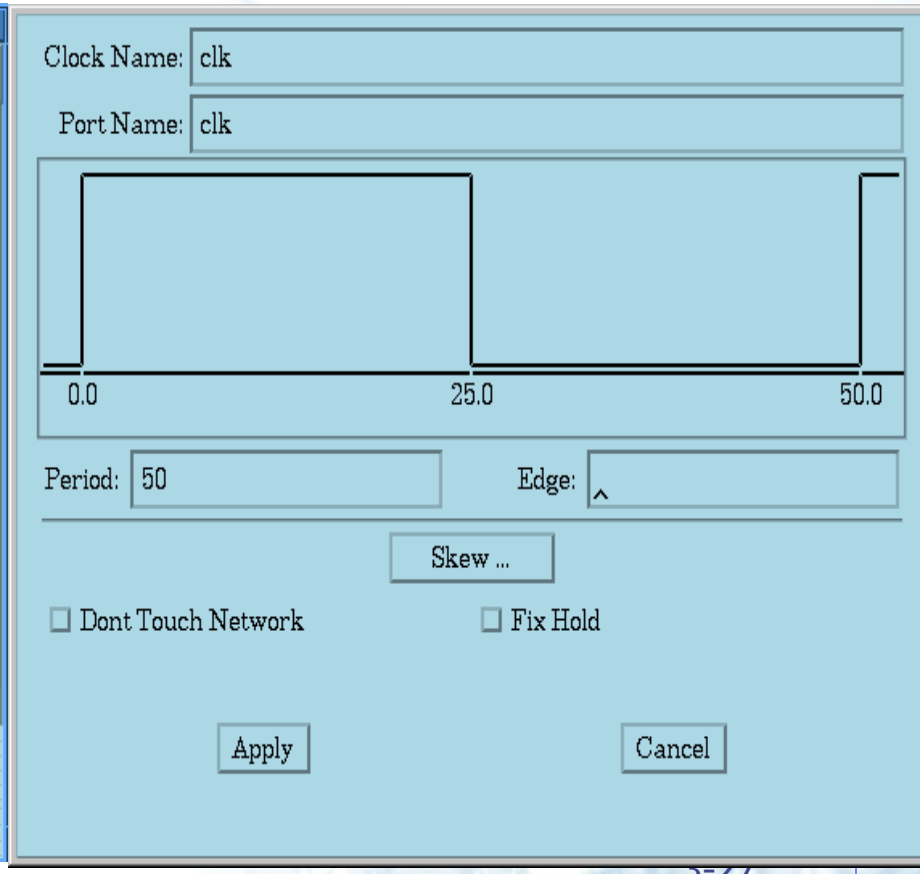
Minimum delay constraint

```
dc_shell> set_max_delay 5.0 -from start_clock -to sec_msb_led
```

Sequential Circuit - Specify Clock (1/2)

- Select clock port
- Attributes/Clocks/Specify





Sequential Circuit - Specify Clock (2/2)

- `create_clock` : define your clock's waveform & respect the set-up time requirements of all clocked flip-flops

```
dc_shell> create_clock "clk" -period 50 -waveform {0 25}
```

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

```
dc_shell> set_fix_hold clk
```

- `set_dont_touch_network` : do not re-buffer the clock network

```
dc_shell> set_dont_touch_network clk
```

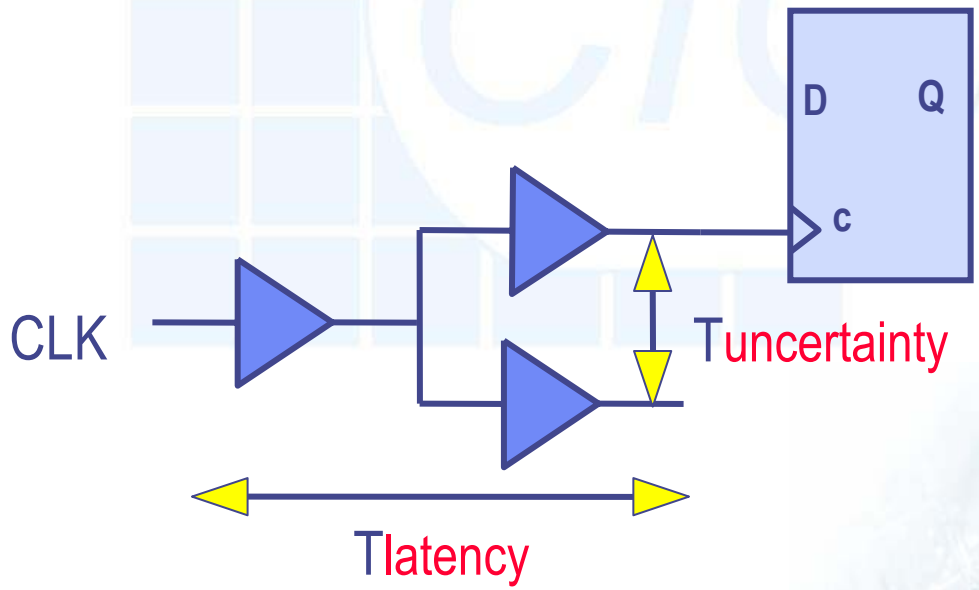
Clock Tree Modeling

- Two parameters to model:
  - Specify clock network latency

    ```
    set_clock_latency -rise tr -fall tf find (clock, CLK)
    ```

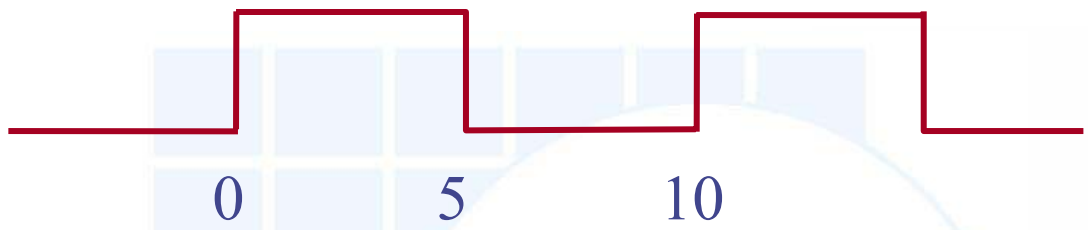
  - Specify uncertainty (skew) of clock networks

    ```
    set_clock_uncertainty -rise tp -fall tm  find (clock,
       CLK)
    ```



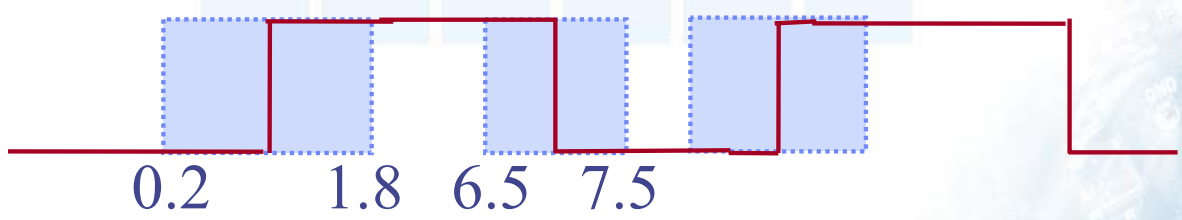
Clock Tree Modeling Example

```
create_clock -period 10 -waveform {0 5} find (port CLK)
```




```
set_clock_latency -rise 1 -fall 2 find (clock CLK)
```

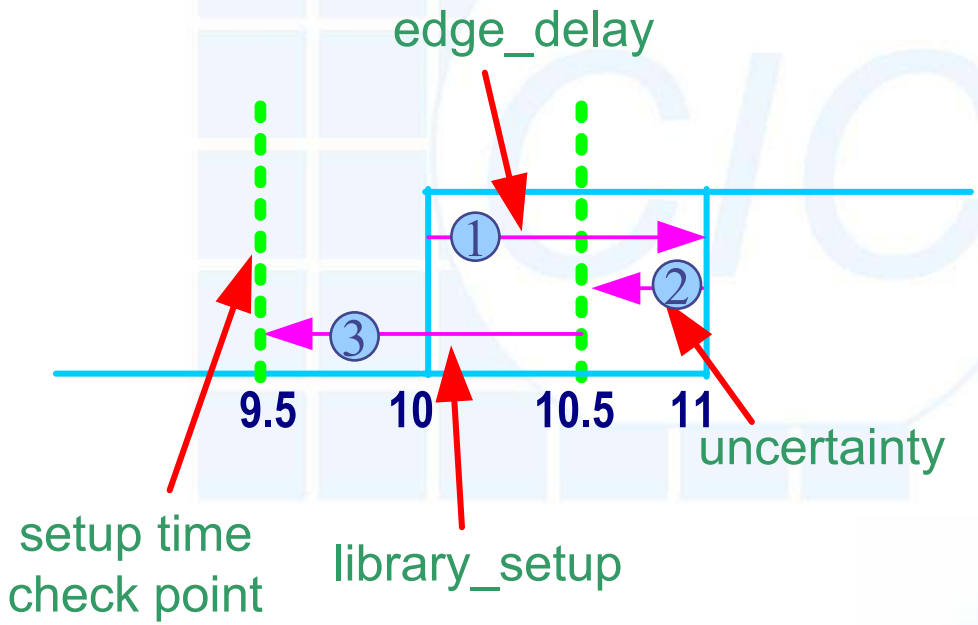

```
set_clock_uncertainty -rise 0.8 -fall 0.5 find (clock CLK)
```



Effect of Clock Tree Modeling on Setup Time

Assumed library (Flip Flop) setup time requirement = **1ns**

```
create_clock -period 10 -waveform {0 5} find (port CLK)
set_clock_latency -rise 1 -fall 2 find (clock CLK)
set_clock_uncertainty -rise 0.5 -fall 0.7  find (clock CLK)
```

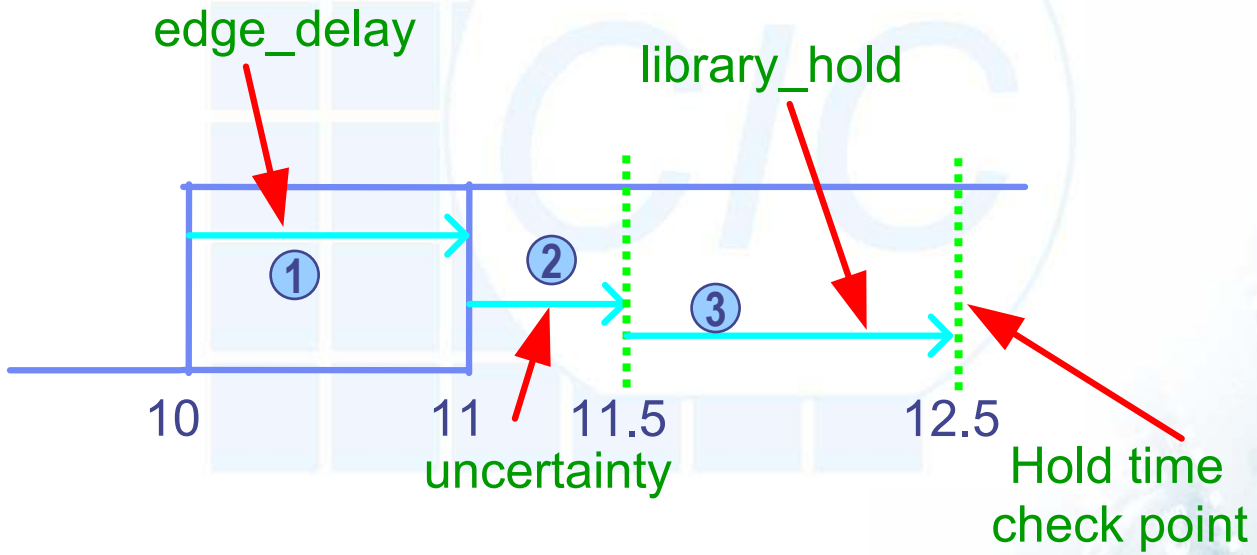


Setup time check = (clock_edge + edge_delay – uncertainty – lib_setup)

Effect of Clock Tree Modeling on Hold Time

Assumed library (Flip Flop) hold time requirement= **1ns**

```
create_clock -period 10 -waveform {0 5} find (port CLK)
set_clock_latency -rise 1 -fall 2 find (port CLK)
set_clock_uncertainty -rise 0.5 -fall 0.8  find (port CLK)
```

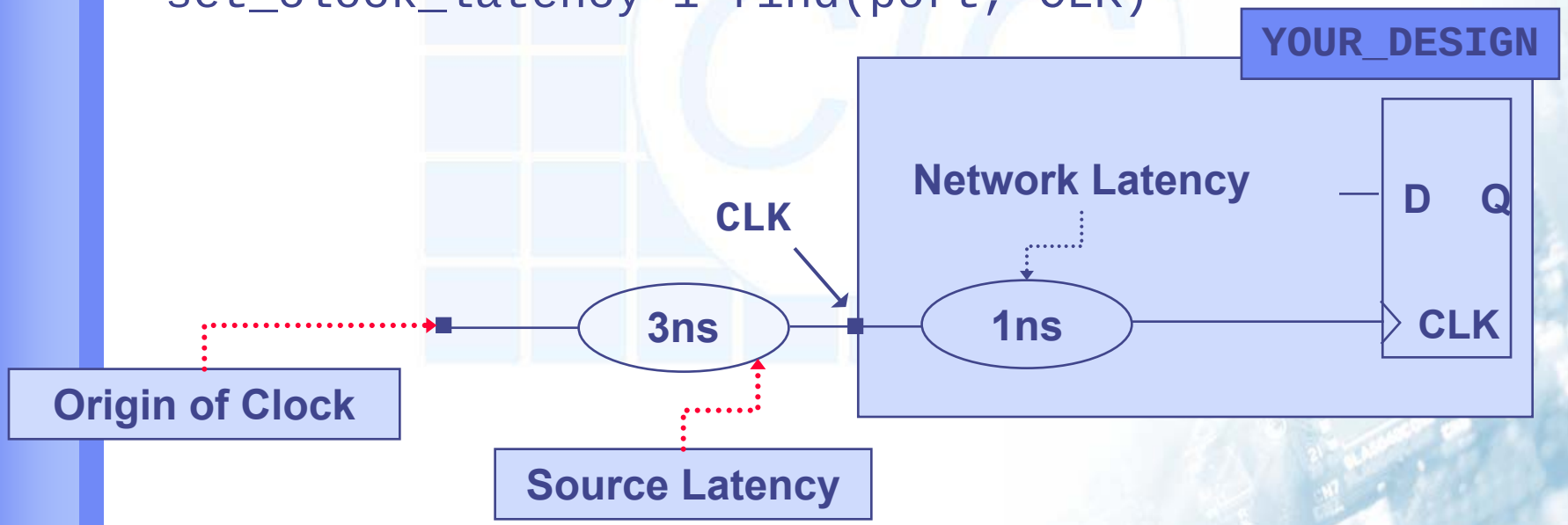


Hold time check = (clock_edge + edge_delay + uncertainty + lib_hold)

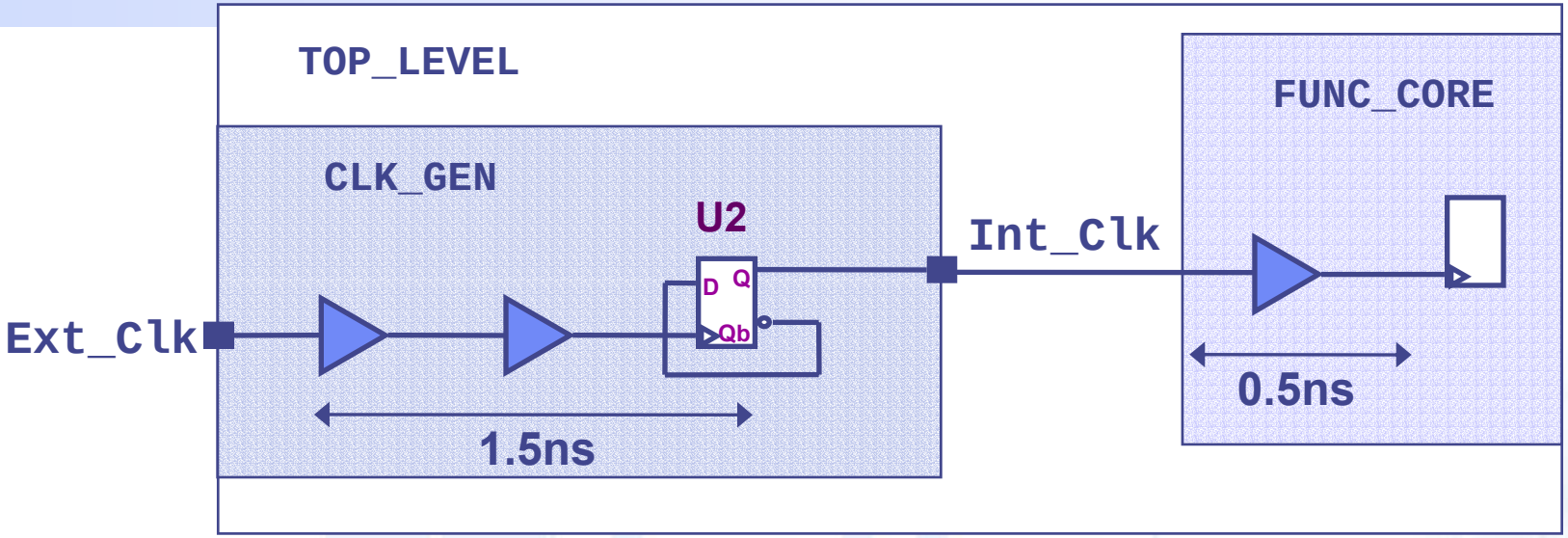
Model Source Latency

- *Source latency* is the propagation time from the actual clock origin to the clock definition point in the design

```
create_clock -period 10 find(port, CLK)
set_clock_latency -source 3 find(port, CLK)
set_clock_latency 1 find(port, CLK)
```



Derived Clocks



Method I

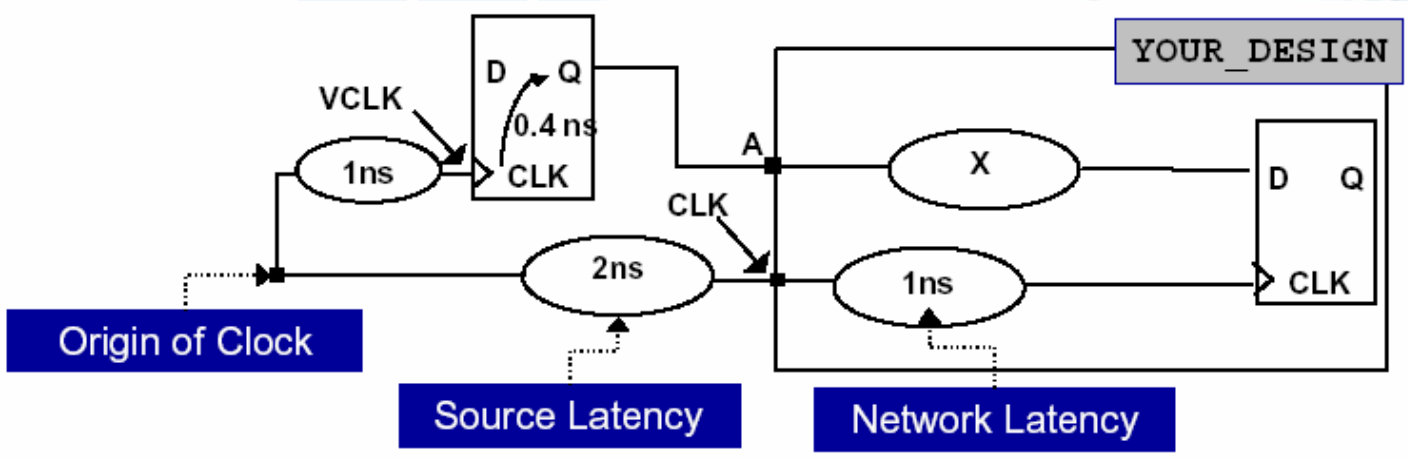
```
create_clock -period 50 find(port, Ext_Clk)
create_clock –name Int_Clk –per 100 find (pin, CLK_GEN/U2/Q)
set_clock_latency -source 1.5 find (pin, CLK_GEN/U2/Q)
set_clock_latency 0.5 find (pin, CLK_GEN/U2/Q)
```

Method II

```
create_clock -period 50 find(port, Ext_Clk)
create_generatd_clock –name Int_Clk  -source Ext_Clk \
 –divide_by 2 find (pin, CLK_GEN/U2/Q)
set_clock_latency -source 1.5 find (pin, CLK_GEN/U2/Q)
set_clock_latency 0.5 find (pin, CLK_GEN/U2/Q)
```

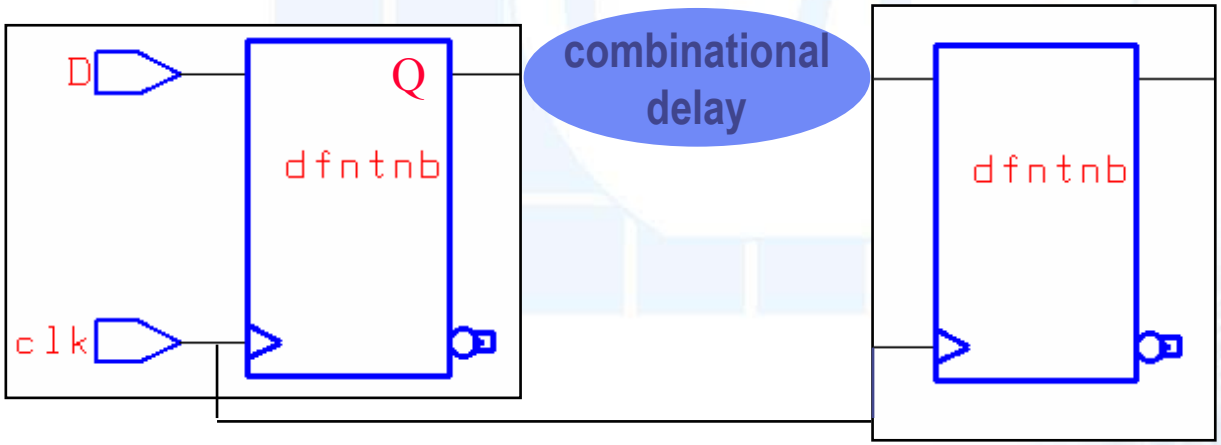
External Clock Delay (example)

```
current_design YOUR_DESIGN
create_clock -p 10 find(port, CLK)
create_clock -p 10 -name VCLK   /* Virtual Clock*/
/*Virtual Clock doen't clock any sequential devices with
   current design*/
set_clock_latency -source 2 find(clock, CLK)
set_clock_latency -source 1 find(clock,VCLK)
set_clock_latency 1 find(clock, CLK)
/* set_propagated_clock all_clocks()*/ /*For post-layout
   Synthesis*/
set_input_delay 0.4 -clock VCLK find(port, A)
```



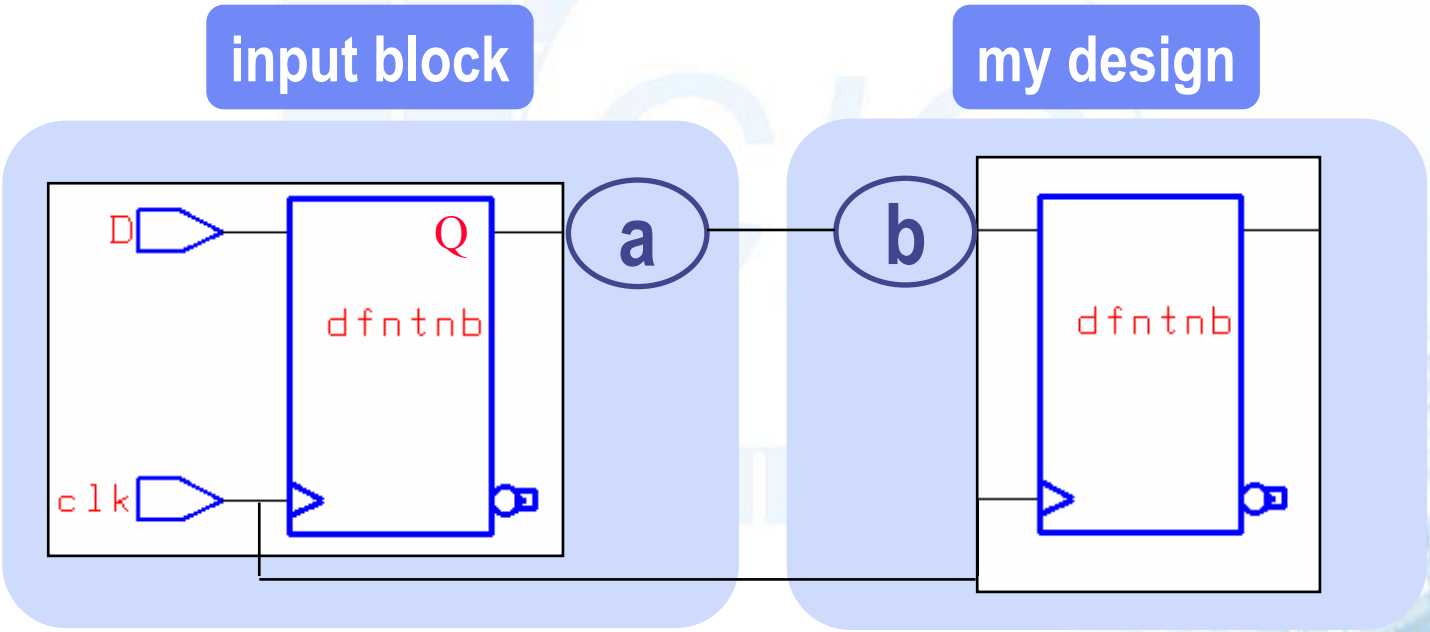
Sequential Circuit

- Sequential circuits are usually constrained by clock specify
- Clock-cycle >= $DFF_{clk\text{-}Qdelay}$ + (Comb. Delay) + DFF_{setup}



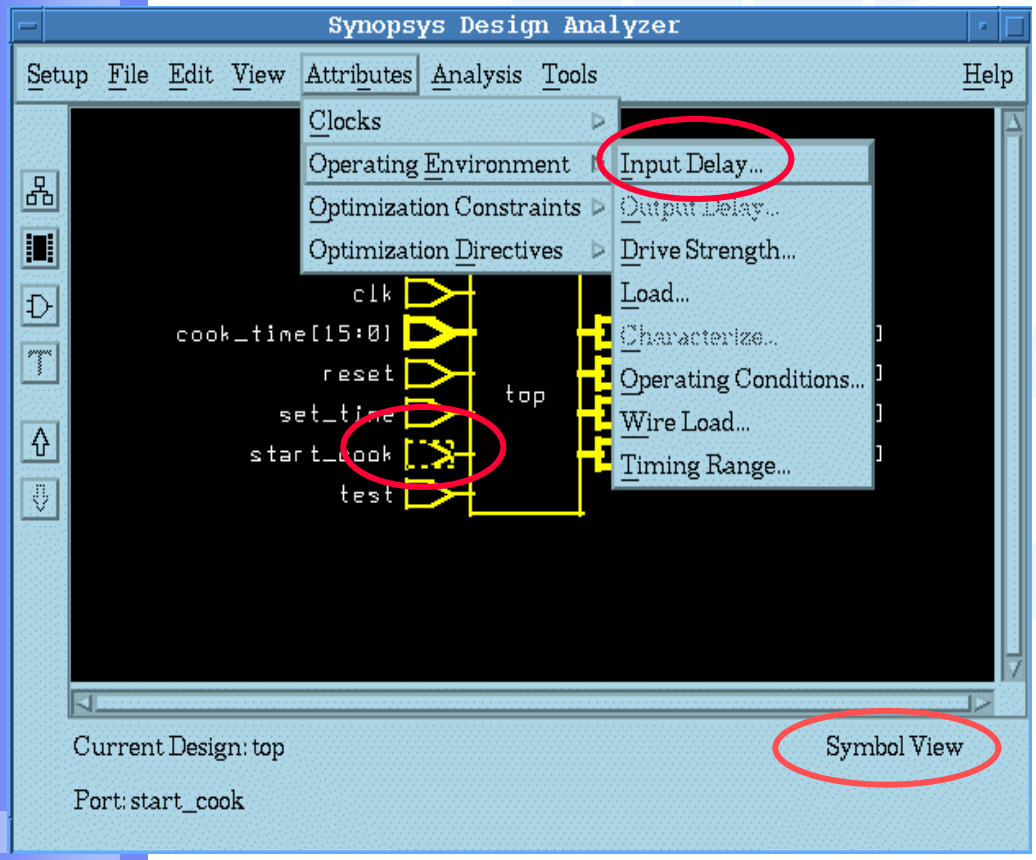
Input Delay Model

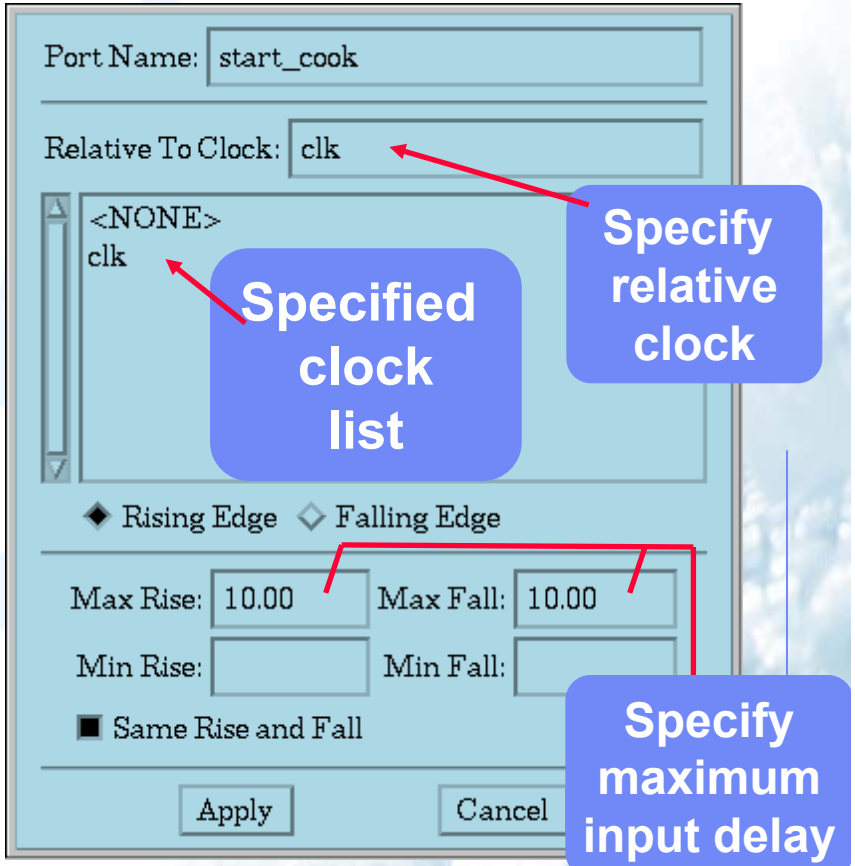
- Clock-cycle >= $DFF_{clk\text{-}Qdelay}$ + a + b + DFF_{setup}
- Input delay = $DFF_{clk\text{-}Qdelay}$ + a



Setting Input Delay (1/3)

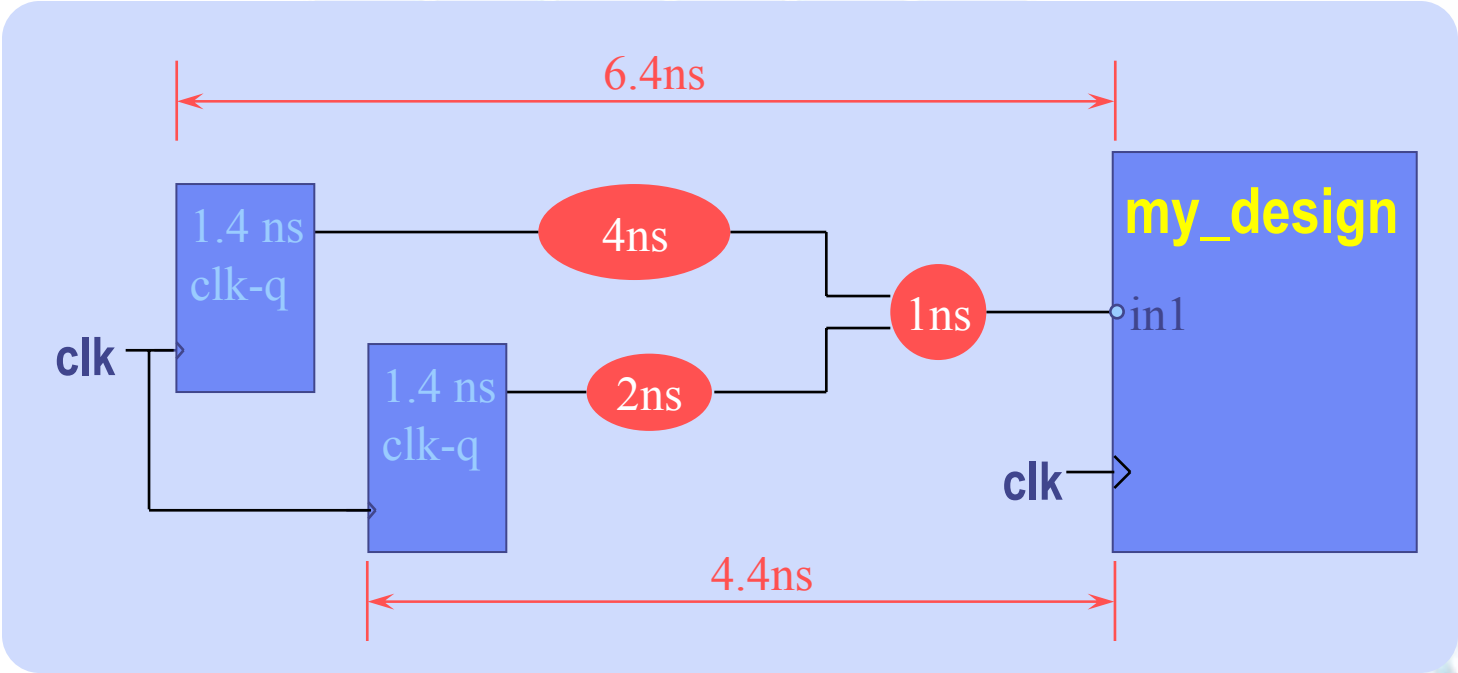
- Select input ports
- Attributes/Operating Environment/Input Delay





Setting Input Delay (2/3)

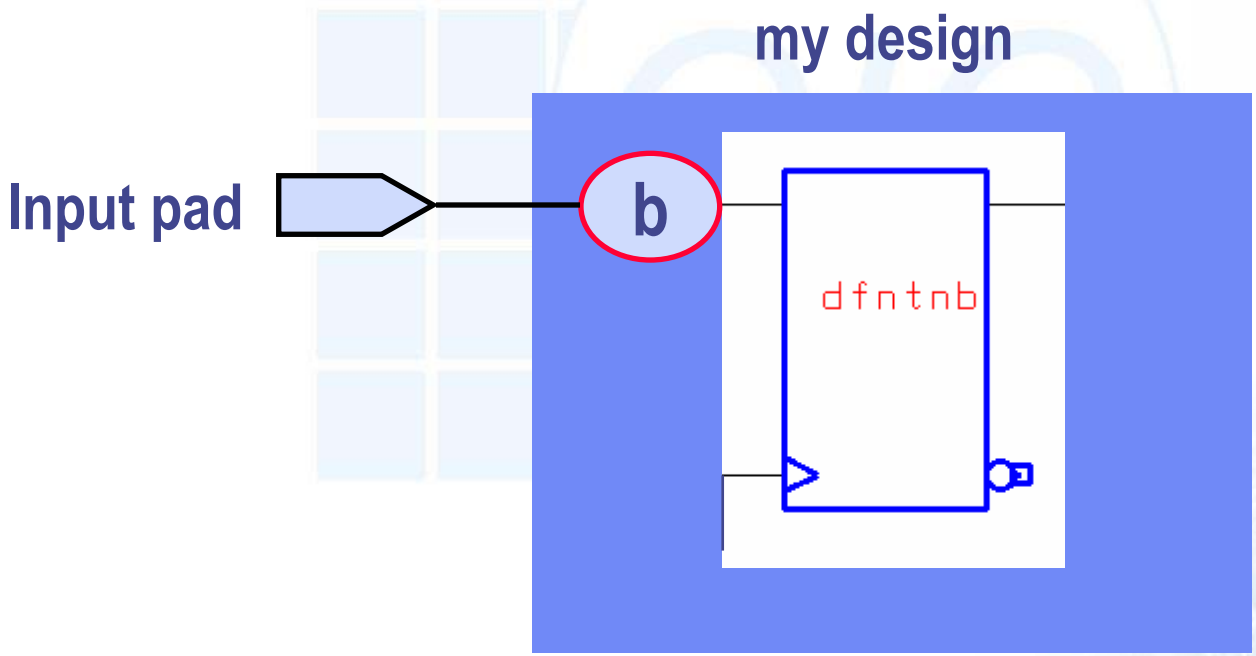
- Example




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

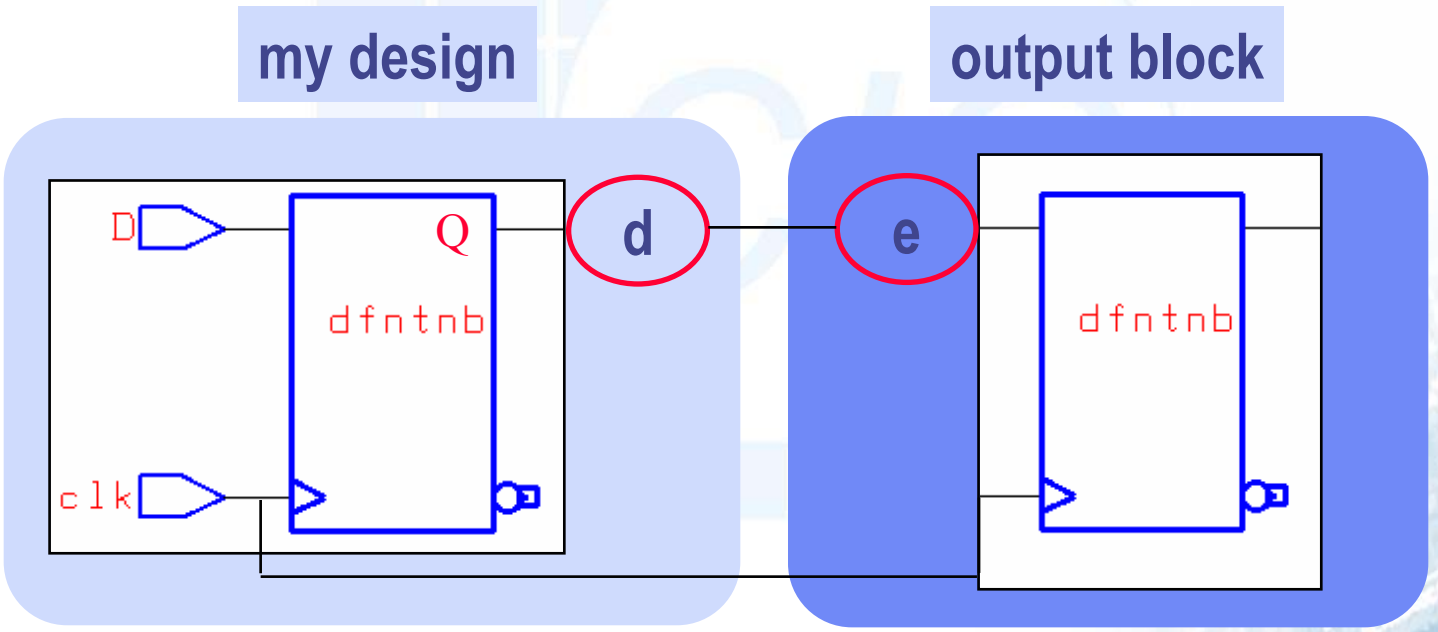
Setting Input Delay (3/3)

- If inputs of your design are input pads (top level) then set the input delay to an appropriate value. (reference to data sheet or use *characterize* command)



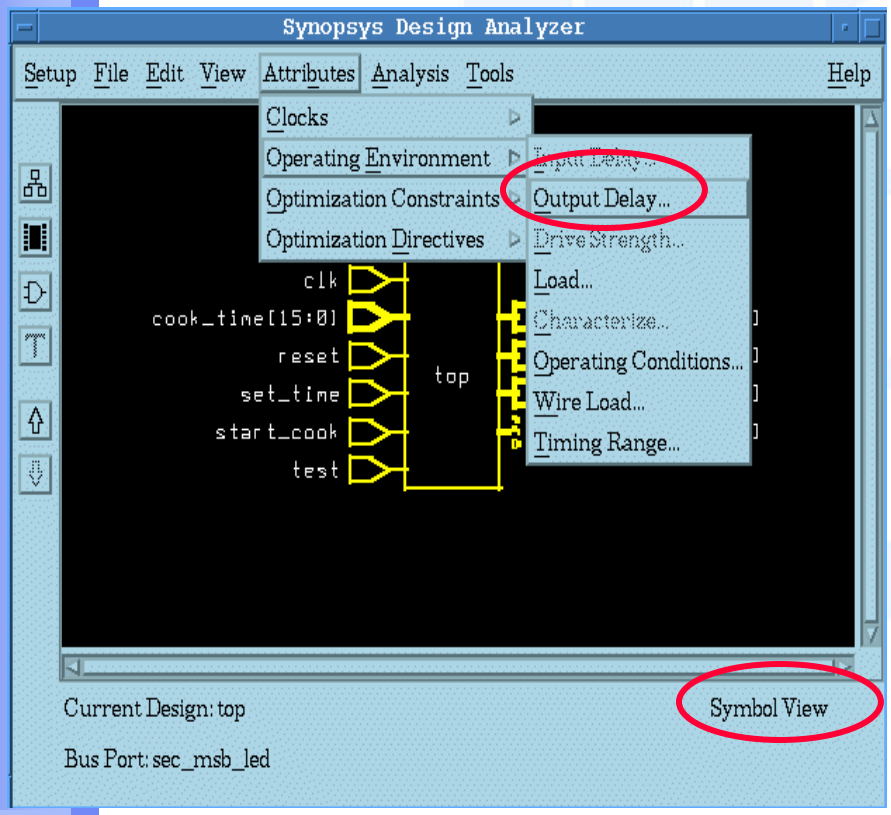
Output Delay Model

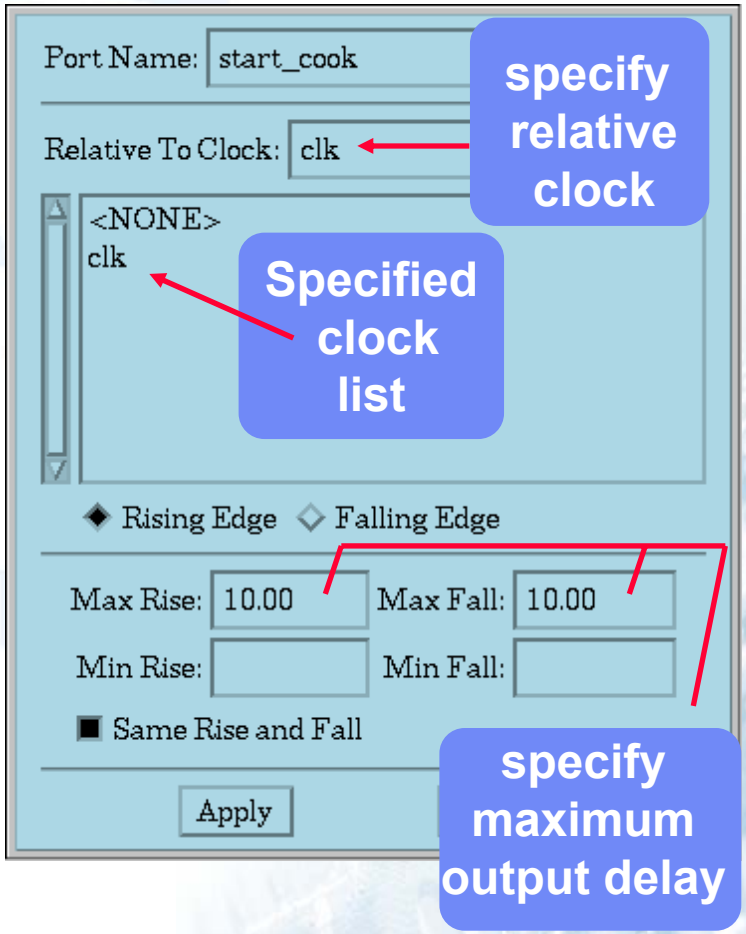
- Clock-cycle >= $DFF_{clk\text{-}Qdelay}$ + d + e + DFF_{setup}
- Output delay = e + DFF_{setup}



Setting Output Delay (1/3)

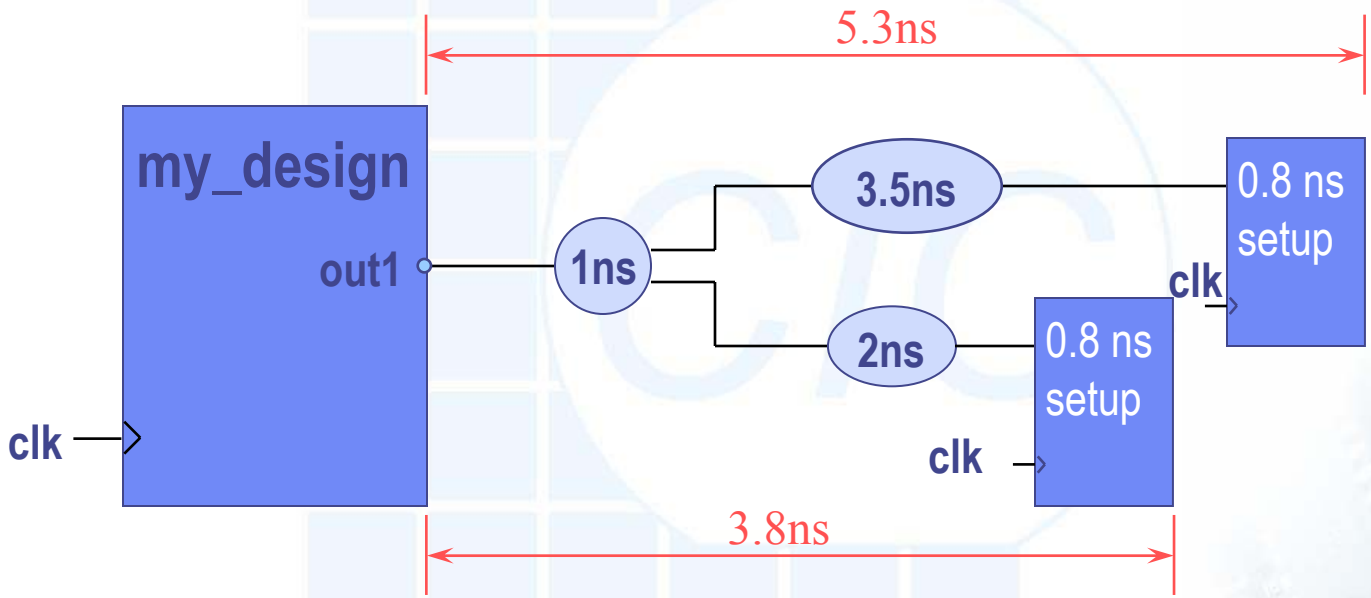
- Select output ports
- Attributes/Operating Environment/Output Delay





Setting Output Delay (2/3)

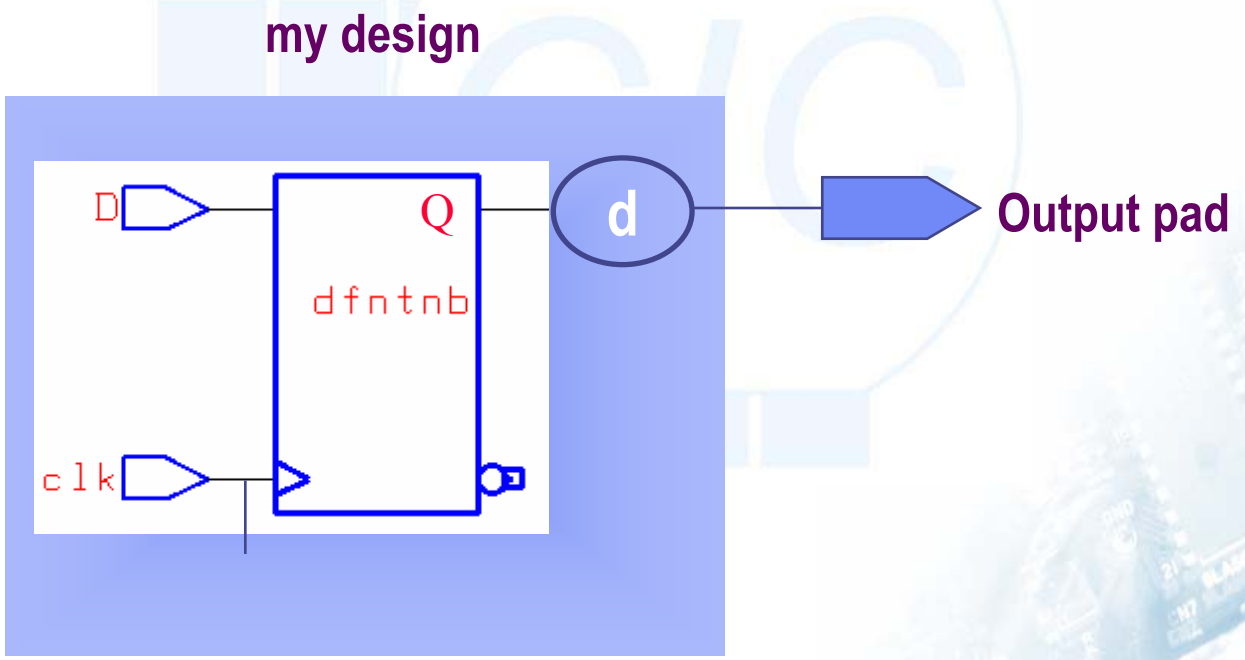
- Example



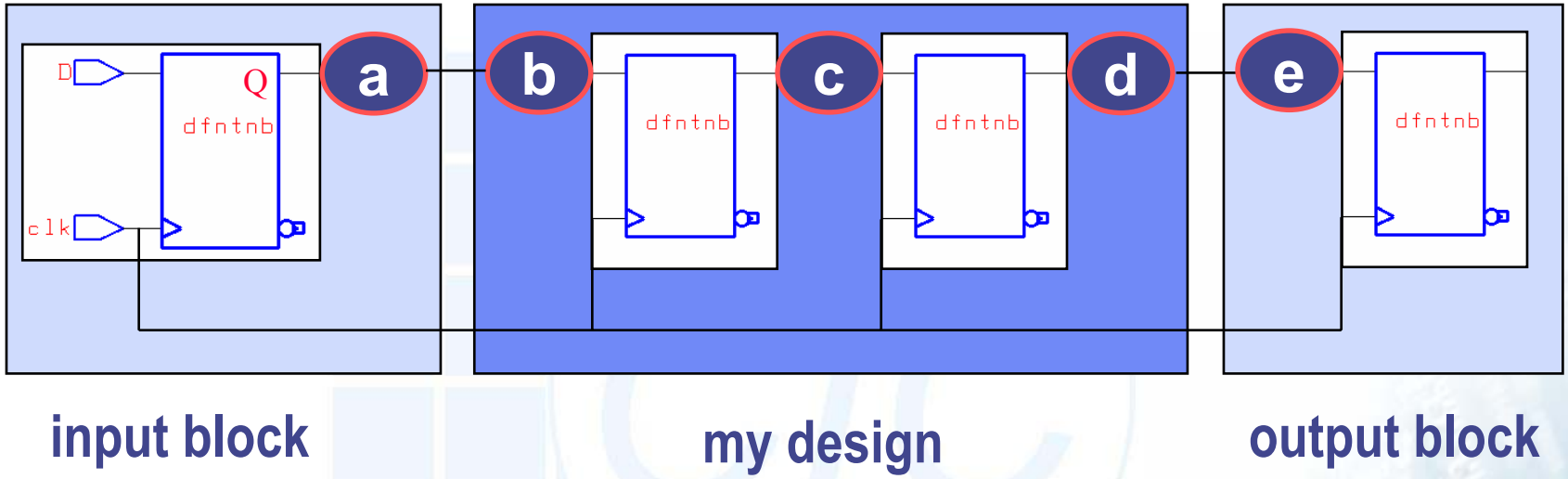

```
dc_shell>set_output_delay -clock clk -max 5.3 out1
```

Setting Output Delay (3/3)

- If outputs of your design are connected to output pads (top level), set the output delay to an appropriate value. (reference to data sheet or use `characterize` command)



What Have We Modeled ?

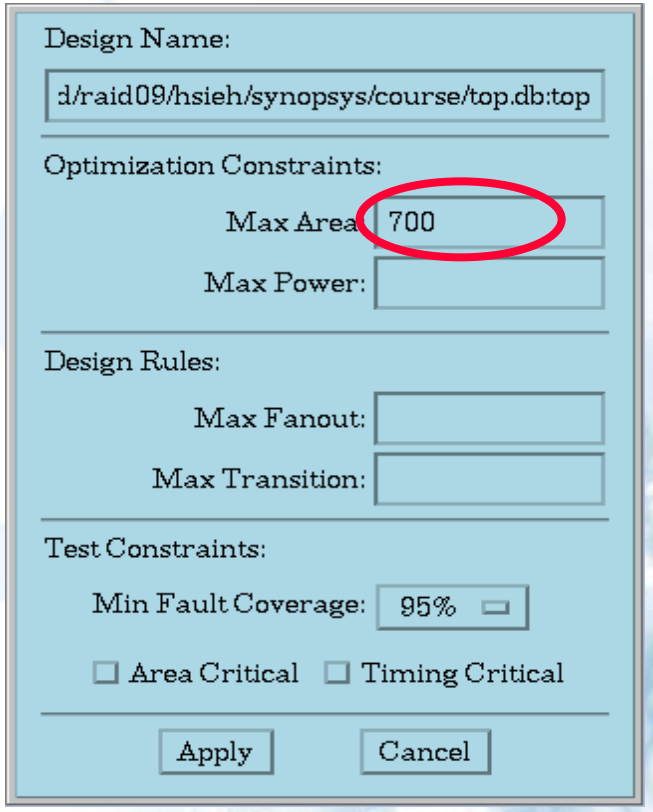


- Assume clock cycle = p
- Input delay = a ; a + b < p
- Output delay = e ; d + e < p

Setting Area Constraint

- Attributes/Optimization Constraints/Design Constraints
- If you only want to concern the area, but don't care the timing. You can use the following constraint script
  - `remove constraints -all`
  - `set max_area 0`
  - `compile -effort medium`

Area unit :
(1) Equivalent gate counts
(2) *umxum*
(3) Transistors



Design Rule Constraints

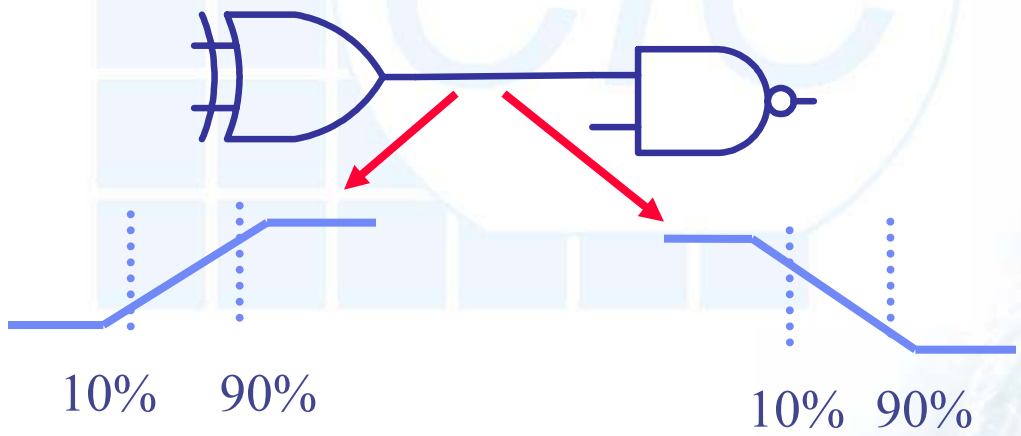
- Design rules cannot be violated at any cost, even if it will violate the timing and area goal

- Three kinds of design rule constraints are set:
  1. `set_max_transition`
  2. `set_max_fanout`
  3. `set_max_capacitance`

Setting Maximum Transition

- `set_max_transition`
  - Set a maximum transition time on ports or design.
  - Example:

```
set_max_transition 5 all_inputs( )
set_max_transition 3 all_outputs( )
```

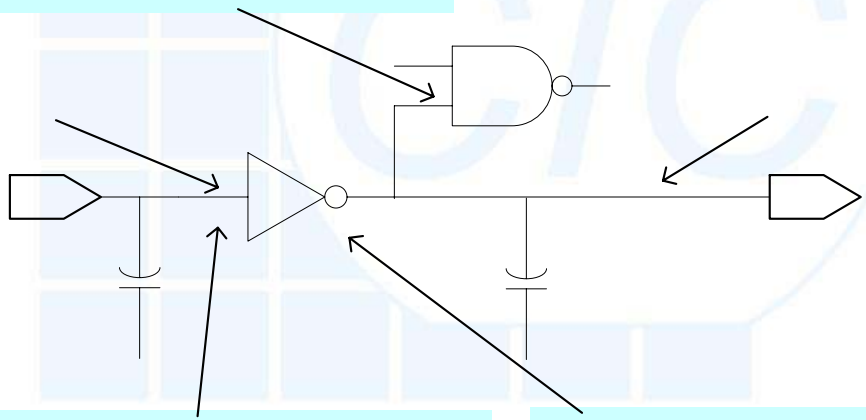


Rising edge on a signal

Falling edge on a signal

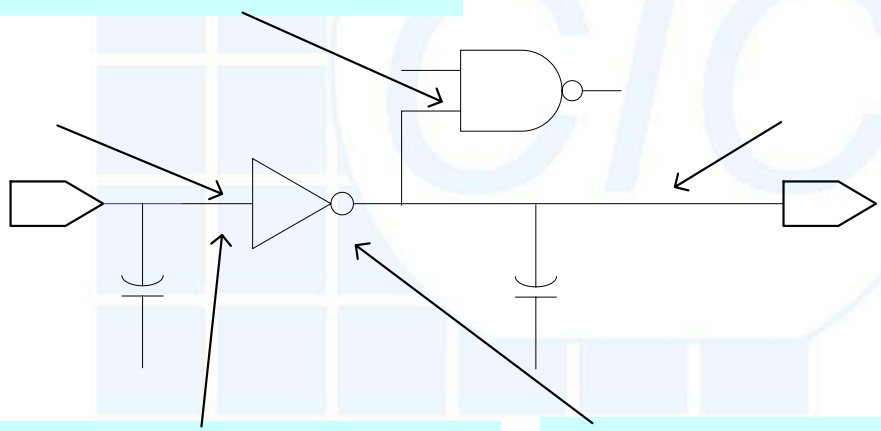
Calculating Maximum Transition Time

- Transition Time = Drive (resistance) * Max_Cap
  = Drive (resistance) * Load ($\sum C_{pins} + C_{wireload}$)



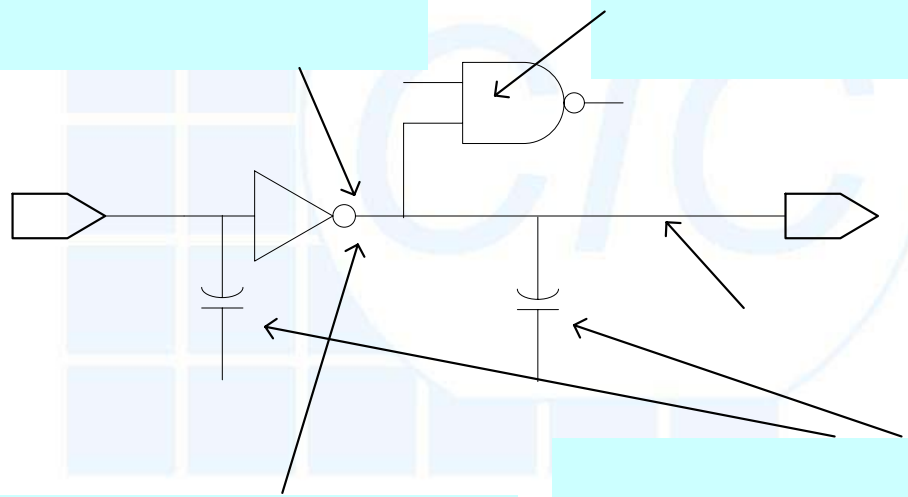
Calculating Maximum fanout_load

- Maximum faout_load = Σ fanout_loads
  - `set_max_fanout value object` (object are input ports or design)



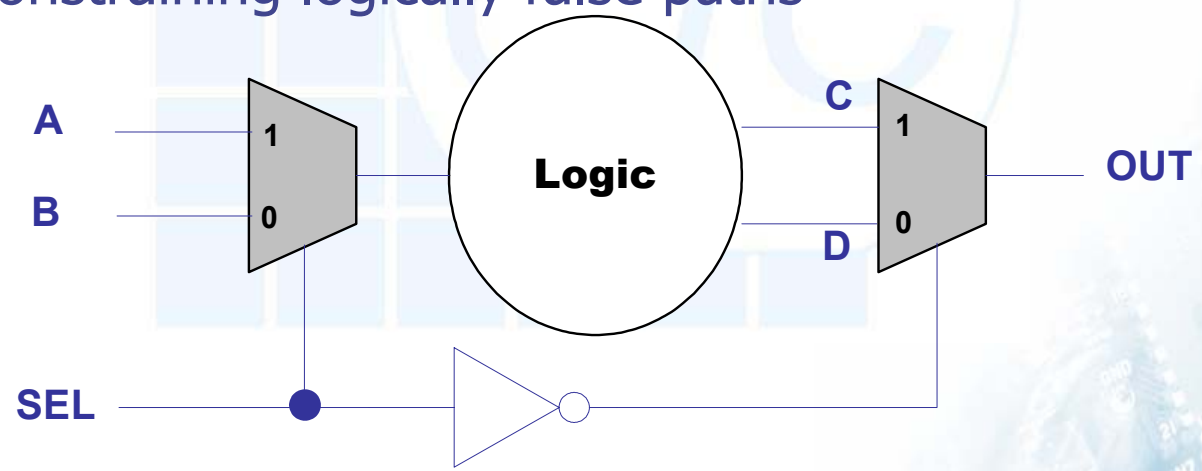
Calculating Maximum Capacitance

- Maximum capacitance = Load($\sum C_{pins} + C_{wireload}$)
  - `set_max_capacitance capacitance_value object`



False Path

- A false path is a timing path that cannot propagate a signal, or a path we wish to ignore timing constraints.
- The `set_false_path` can be used to disable timing-based synthesis on a path-by-path basis
- It is useful for:
  - Constraining asynchronous paths
  - Constraining logically false paths



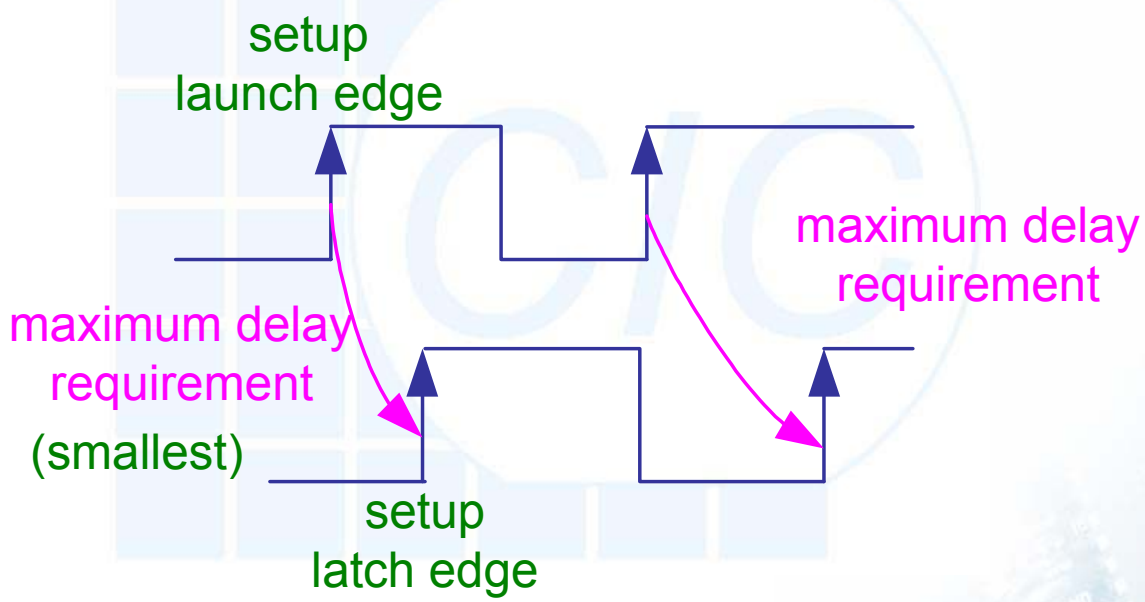

```
set_false_path -from {A} -through {C} -to {OUT}
set_false_path -from {B} -through {D} -to {OUT}
```

Constraining Multi-frequency Designs

- Flip flops
  - A single active edge both launches and captures data.
- Latches
  - The open edge launches data and the close edge latches data.
- Assume all clocks are synchronized

Setup Check in Multi-frequency

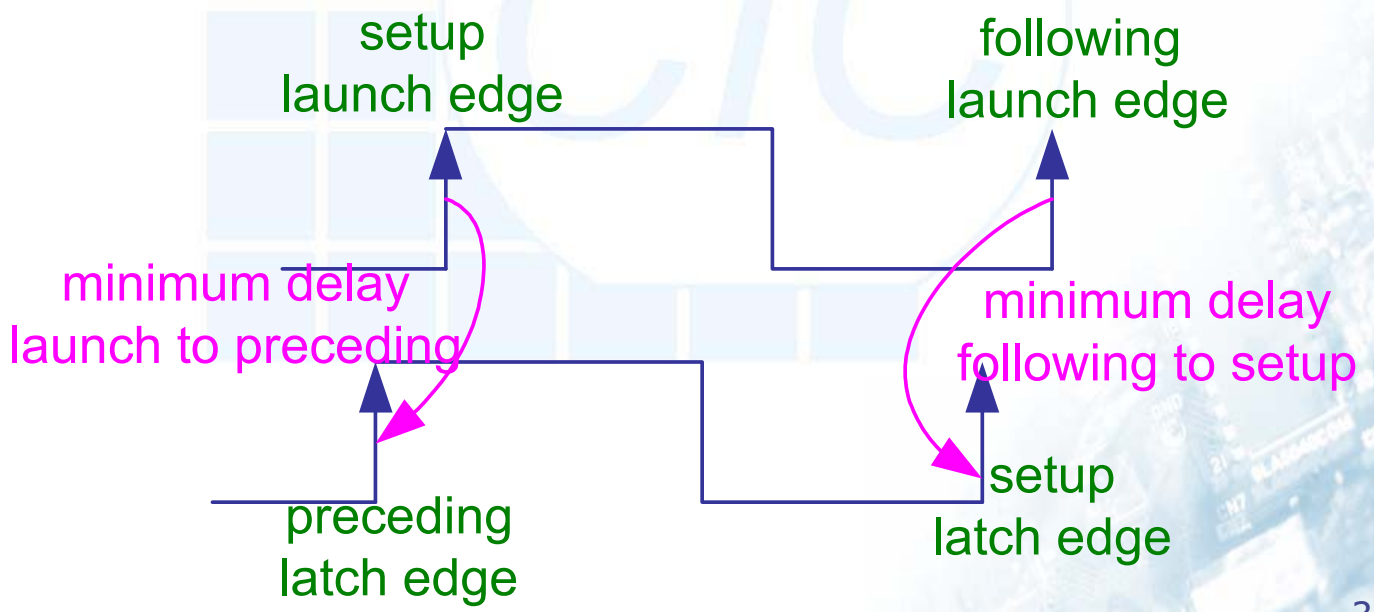
- Determine the smallest maximum delay requirement which satisfies:
  - For every latch edge of the destination clock, find the nearest launch edge that precedes each capture edge



  - Use `set_multicycle_path` or `set_max_delay` / `set_min_delay` to override the default clocking

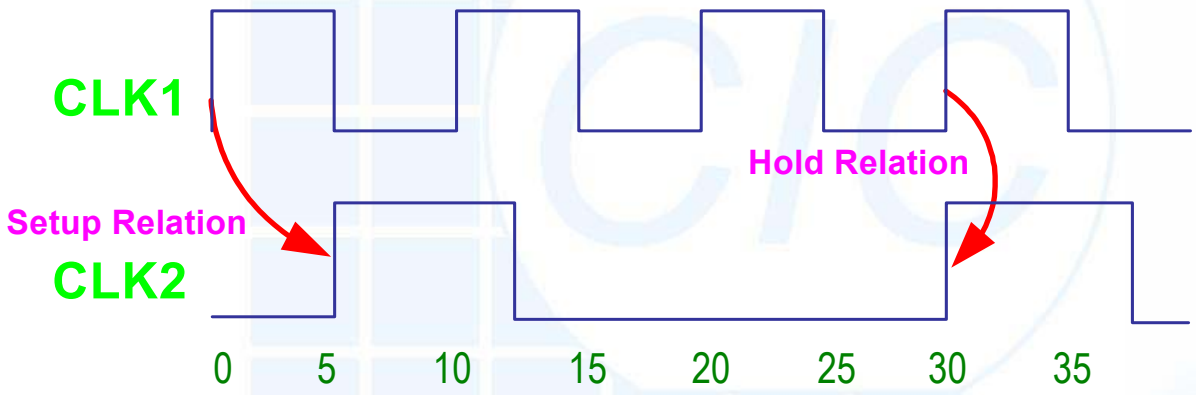
Hold Check in Multi-frequency

- Determine the largest minimum delay requirement which satisfies
  - Data from the source clock edge that follows the setup launch edge must not be latched by the setup latch edge.
  - Data from the setup launch edge must not be latched by the destination clock edge that precedes the setup latch edge.

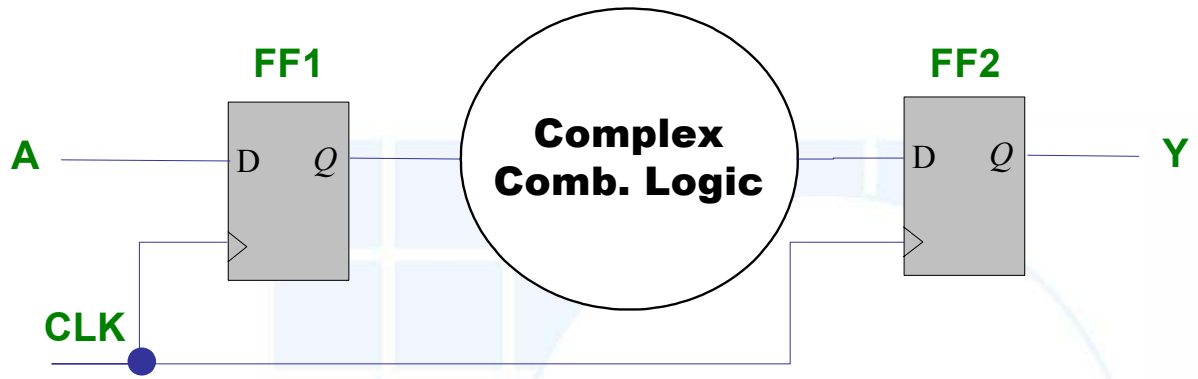


Example

- The most restrictive setup relation is 5 ns (from CLK1 edge at 0 to CLK2 edge at 5)
- The most restrictive hold time is 0 ns (from CLK1 edge at 30 to CLK2 edge at 30)



Multicycle Path



- In some cases, combinational logic delay between two registers may require more than one clock cycle. Such paths should be set as *multicycle* paths.

```
set_multicycle_path 2 -from FF1 -to FF2
```

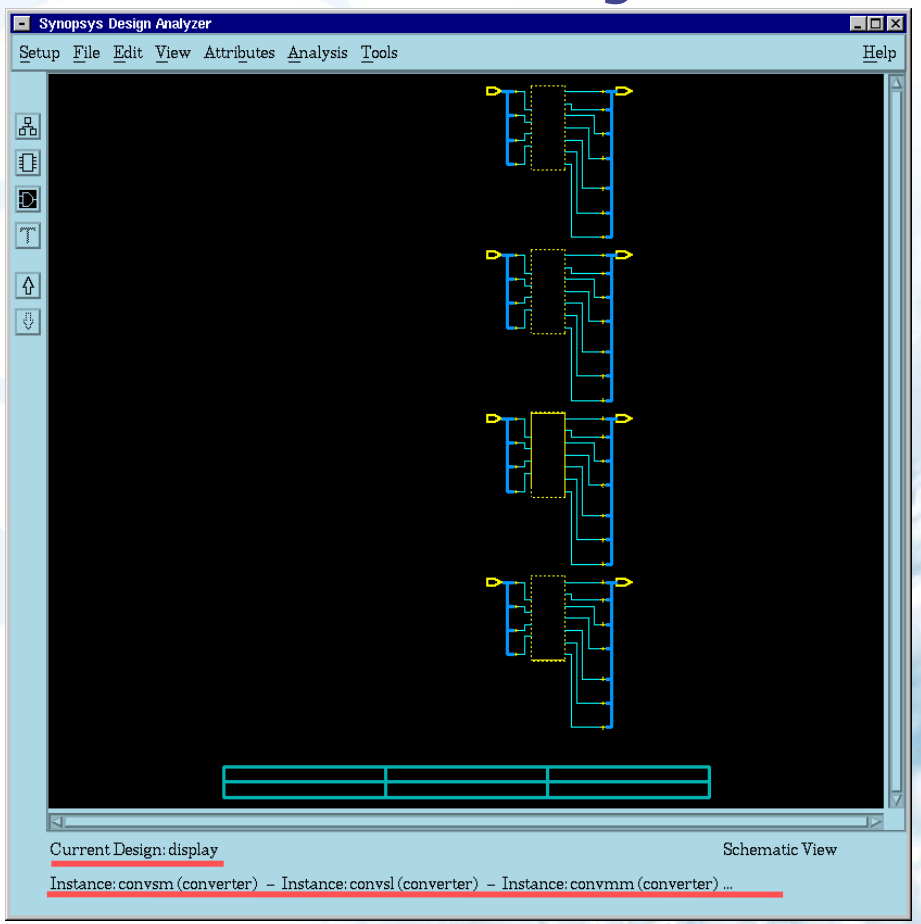
Check Design (1/2)

- After you set up the deign attributes & design constraints, we recommend the next step is to check design.
- Analysis/Check Design
- Maybe you will meet the warning message shown as follow.

```
design_analyzer> Warning: Design 'converter' is instantiated 4 times. (LINT-45)
         Cell 'convml' in design 'display'
         Cell 'convmm' in design 'display'
         Cell 'convsl' in design 'display'
         Cell 'convsm' in design 'display'
1
design_analyzer>
```

Check Design (2/2)

- 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

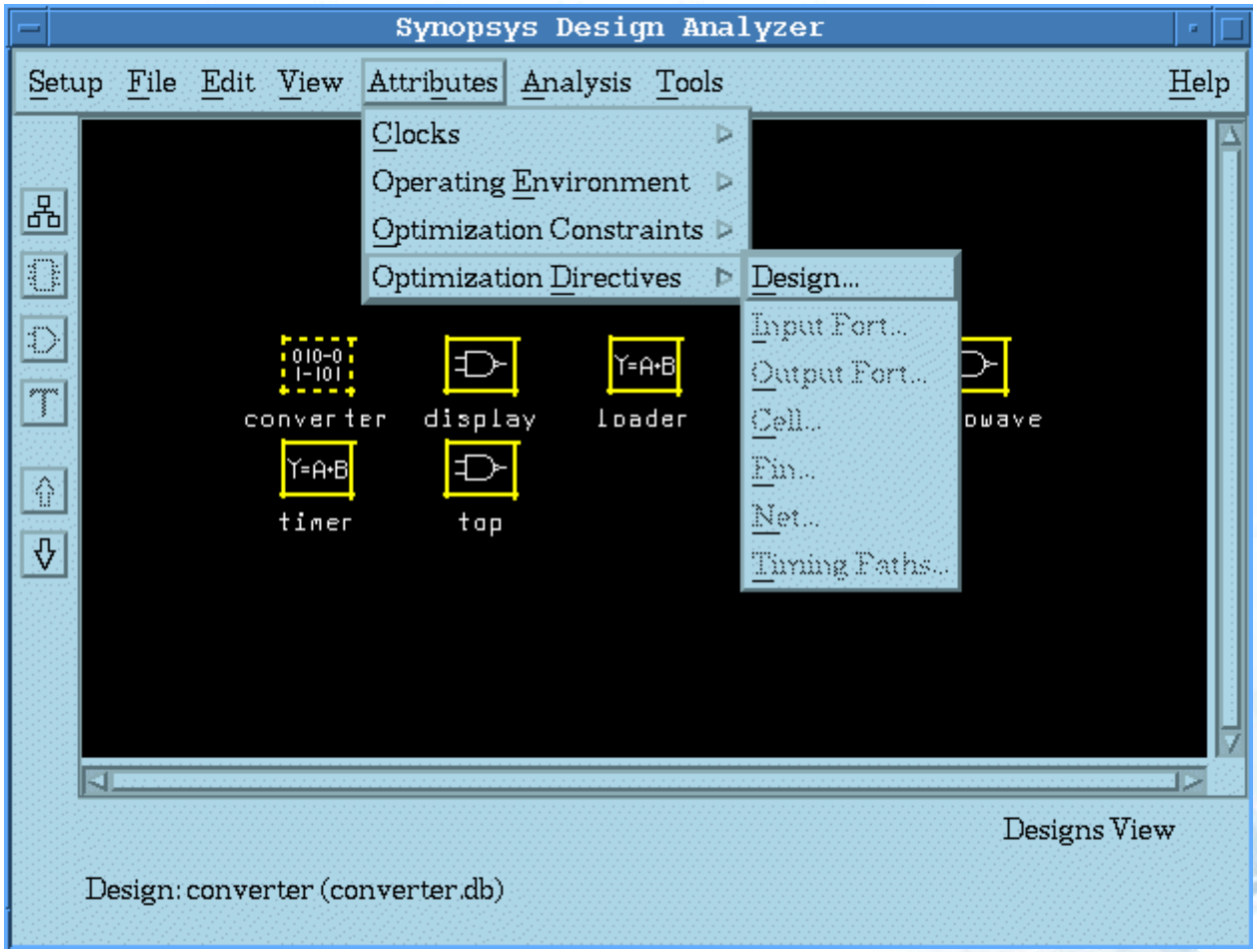


dont_touch (1/3)

- Inhibit re-compile of a lower level design.
- Hierarchy will be maintained.
- A single design representation might be shared.
- Used for blocks that requires little customization.
- During design optimization, the dont_touch block will not be re-optimized.
- If dont_touch is placed on an unmapped design, the design will remain unmapped.

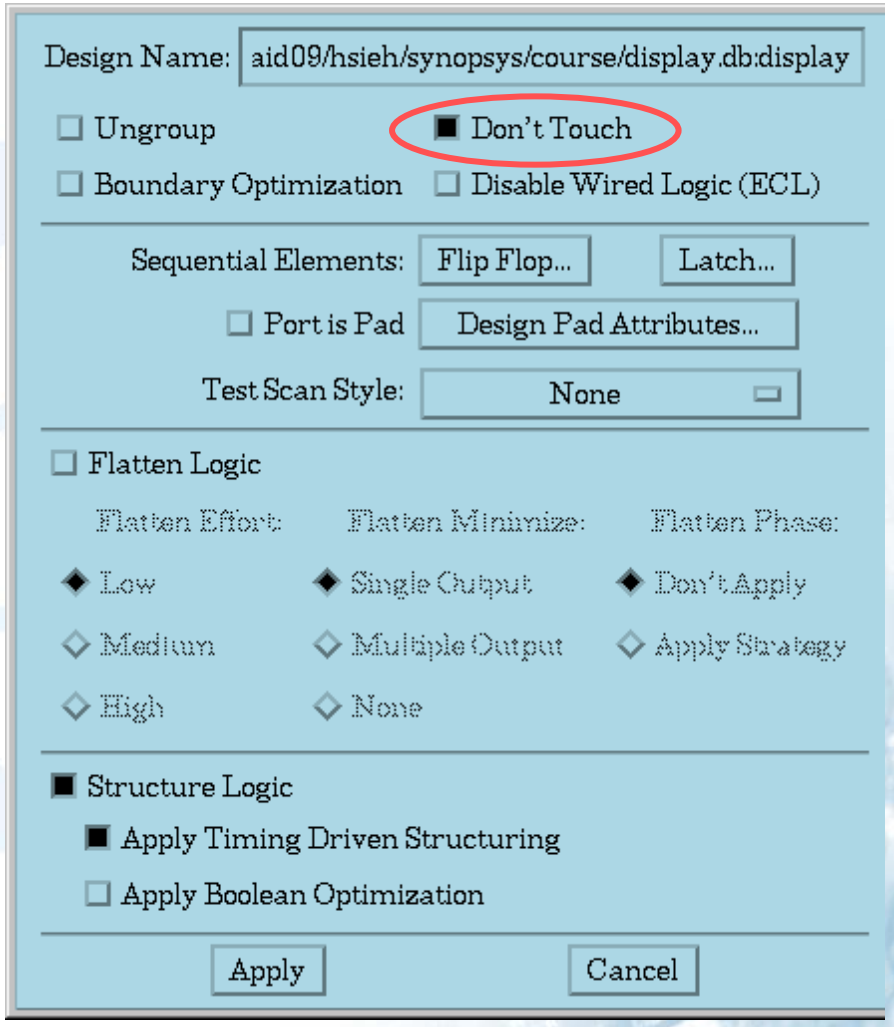
dont_touch (2/3)

- Attributes/Optimization Directives/Design



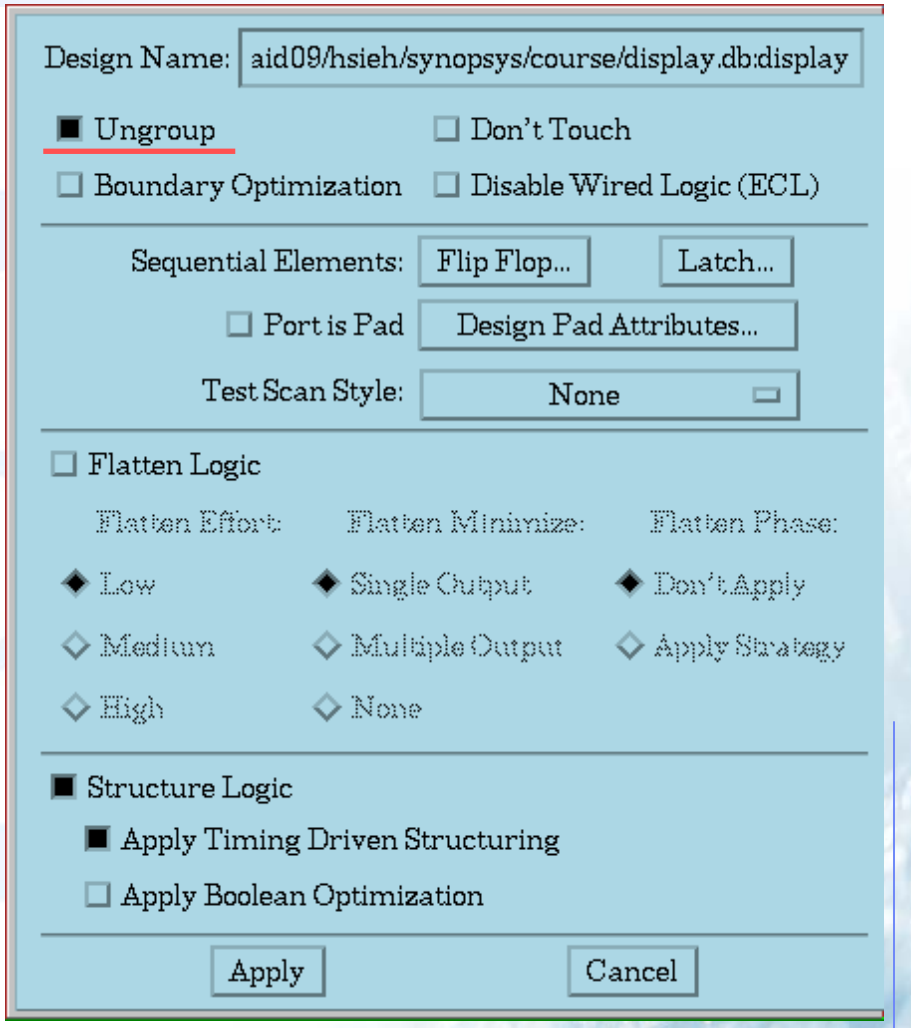
dont_touch (3/3)

- Procedure
  - 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



Ungroup

- Procedure
  - 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 and compile time

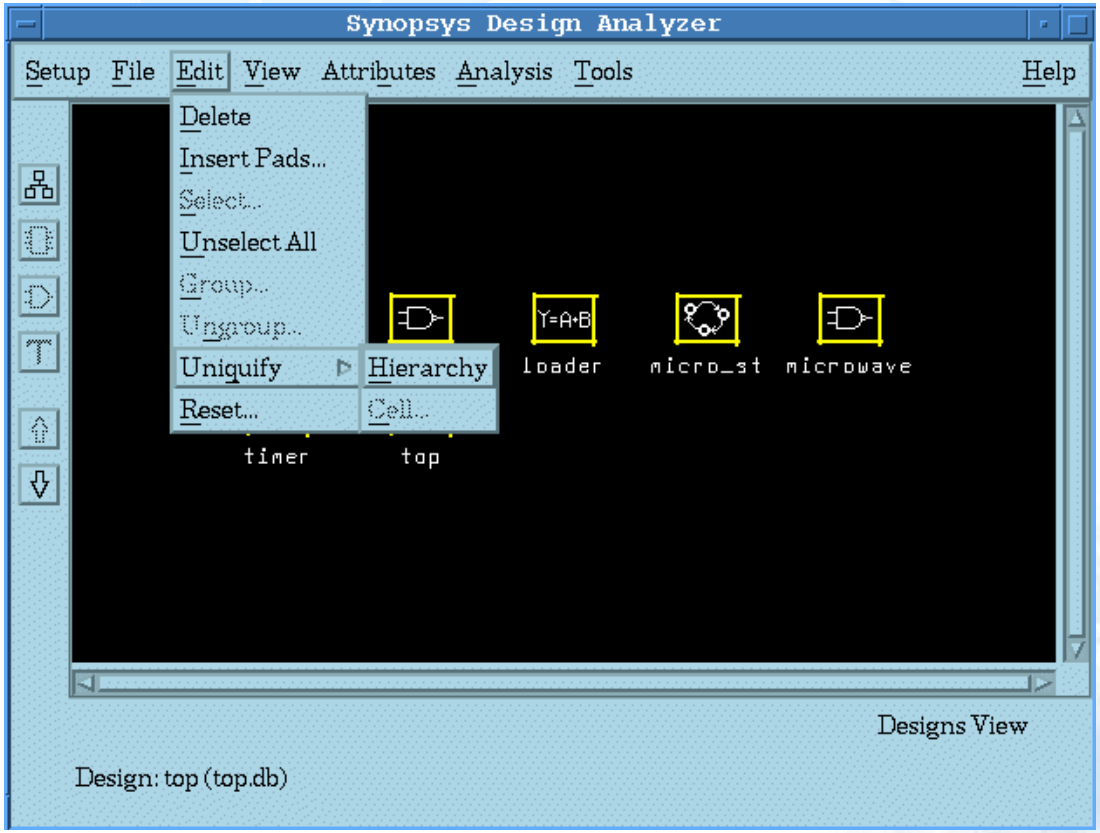


Uniquify (1/3)

- 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 causes longer compile time than compile+dont_touch.

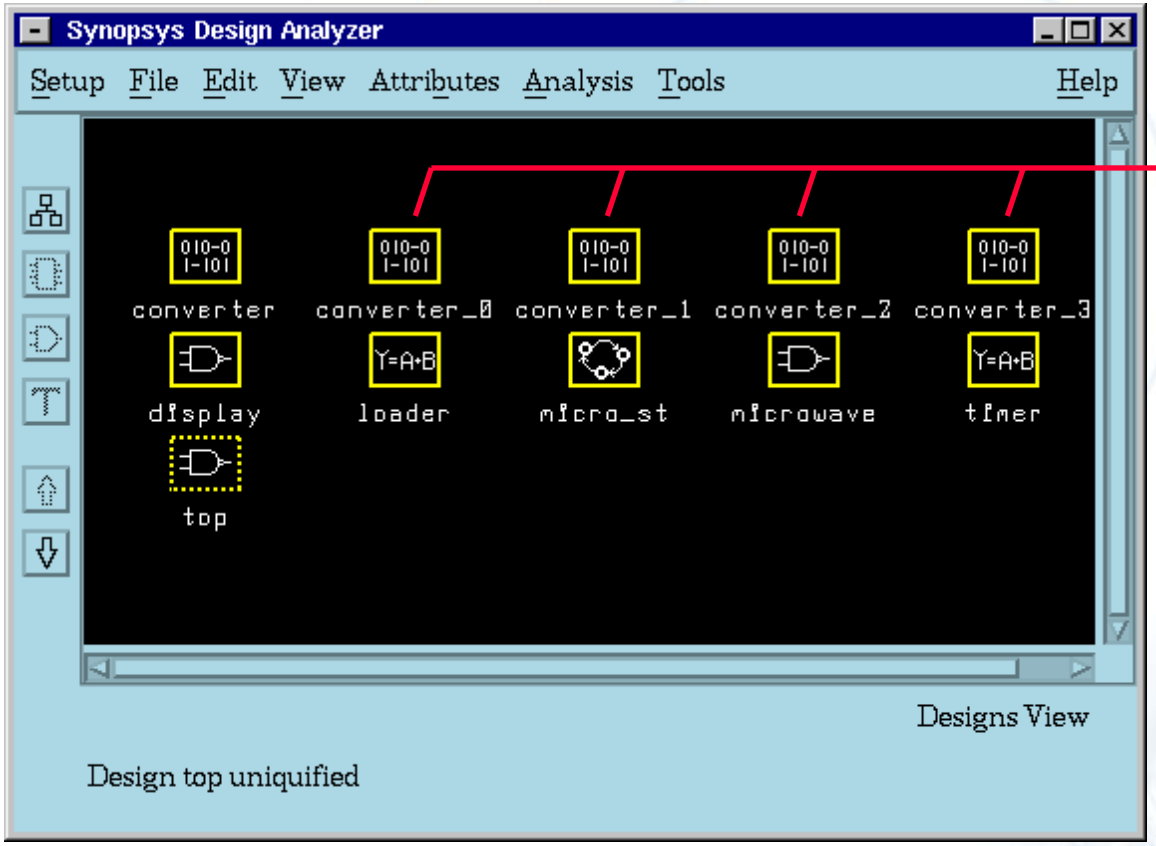
Uniquify (2/3)

- Select the most top design of the hierarchy.
- Edit/Uniquify/Hierarchy



Uniquify (3/3)

- Use the uniquify_naming_style variable to create a new design name, default is **%s_%d**



uniquified instances

Multiple Design Instance (summary)

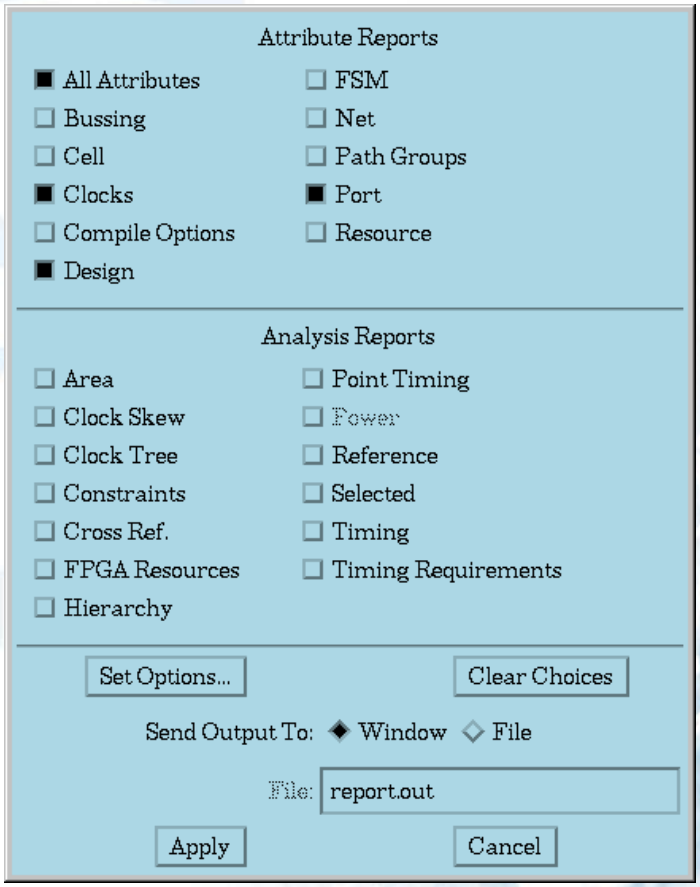
- Use "dont_touch, ungroup, uniquify" to fix it
- The easiest way is uniquify, but needs much memory & compile time.
- If you want to preserve the hierarchy & source sharing, use dont_touch.
- If you want your design to have the **BEST** result, recommend to use ungroup. But it needs the most memory and compile time.

Constraints Priority

1. During the optimization, there exists a constraint priority relationship
   1. Design Rule Constraint
      (max_transition, max_fanout, max_capacitance)
   2. Timing constraint
      (max_delay, min_delay)
   3. Power constraint
   4. Area constraint
- Use set_cost_priority command to modify the order
  - `set_cost_priority [-default] [-delay] [cost_list]`

Check Constraints & Attributes

- Use the following reports to check constraints & attributes before compiling.
- Analysis/Report



Attribute Report

Report Output

```
design_analyzer>
****************************************
Report : Attribute
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 18:11:24 2003
****************************************

Design        Object        Type      Attribute Name            Value
-----------------------------------------------------------------------
top           top           design    hdl_parameters
top           top           design    hdl_canonical_params
top           top           design    hdl_template              top
top           top           design    physical_scale_factor     1000
top           top           design    max_area                  600.000000
top           top           design    wire_load_selection_type  1
top           top           design    wire_load_model_mode      top
```

Show Next Previous Cancel

Area constraint

```
top           cook_time[12]    port    fall_drive          3.000000
top           cook_time[12]    port    fall_wire_drive     3.000000
top           cook_time[13]    port    rise_drive          2.010000
top           cook_time[13]    port    rise_wire_drive     2.010000
top           cook_time[13]    port    fall_drive          3.000000
top           cook_time[13]    port    fall_wire_drive     3.000000
top           cook_time[14]    port    rise_drive          2.010000
top           cook_time[14]    port    rise_wire_drive     2.010000
top           cook_time[14]    port    fall_drive          3.000000
top           cook_time[14]    port    fall_wire_drive     3.000000
top           cook_time[15]    port    rise_drive          2.010000
top           cook_time[15]    port    rise_wire_drive     2.010000
top           cook_time[15]    port    fall_drive          3.000000
top           cook_time[15]    port    fall_wire_drive     3.000000
top           sec_msb_led[0]   port    load                0.019000
top           sec_msb_led[1]   port    load                0.019000
top           sec_msb_led[2]   port    load                0.019000
top           sec_msb_led[3]   port    load                0.019000
top           sec_msb_led[4]   port    load                0.019000
top           sec_msb_led[5]   port    load                0.019000
top           sec_msb_led[6]   port    load                0.019000

1
design_analyzer>
```

Show Next Previous Cancel

In this report, you can check the input drive strength & the output load

Clock Report

Report Output

```
****************************************
Report : clocks
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 18:15:44 2003
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock

Clock          Period   Waveform          Attrs     Sources
-------------------------------------------------------------------------
clk             50.00   {0 25}            d f       {clk}
-------------------------------------------------------------------------
1
design_analyzer>
```

Duty Cycle

Show | Next | Previous | Cancel

Design Report

Report Output

```
design_analyzer>
****************************************
Report : design
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 18:18:10 2003
****************************************

Design allows ideal nets on clock nets.

Library(s) Used:
    gtech (File: /usr/synopsys/synthesis/cur/libraries/syn/gtech.db)

Flip-Flop Types:
    No flip-flop types specified.

Latch Types:
    No latch types specified.

Operating Conditions:
 Name          Library    Calc_mode Process
                                        Temp    Volt    Interconnect Model
 ------------------------------------------------------------------------
 WCCOM         cb35os142            1.32 100.00    2.70 worst_case_tree
```

Show | Next | Previous | Cancel

Report Output

```
Wire Loading Model:
    Selected manually by the user.
:
Name            :   ForQA
Location        :   cb35os142
Resistance      :   0
Capacitance     :   1
Area            :   0.1
Slope           :   0.01
Fanout   Length   Points Average Cap Std Deviation
-------------------------------------------------------------
     1     0.03
     2     0.04
     3     0.05
     4     0.06
     5     0.07
     6     0.08
     7     0.09
     8     0.10
     9     0.11
    10     0.12
    11     0.13
    12     0.14
    13     0.15
    14     0.16
```

Show | Next | Previous | Cancel

In this report you can check the operating condition and wire load model

Port Report (1/2)

- To get more information, use the dc_shell command
  `dc_shell> report_port -verbose { port_list }`

Command Window

```
                    Input Delay
                   Min            Max        Related    Max
Input Port     Rise    Fall    Rise    Fall   Clock  Fanout
-------------------------------------------------------------------------
clk            --      --      --      --      --      --
start_cook     --      --     10.00   10.00  clk      --
cook_time[15]
               --      --      --      --      --      --
cook_time[14]
               --      --      --      --      --      --
cook_time[13]
               --      --      --      --      --      --
cook_time[12]
               --      --      --      --      --      --
cook_time[11]
               --      --      --      --      --      --
cook_time[10]
               --      --      --      --      --      --
cook_time[9]
               --      --      --      --      --      --
cook_time[8]
```

design_analyzer>

Port Report (2/2)

Command Window

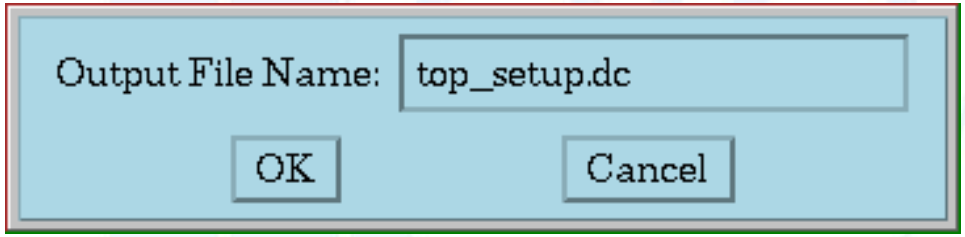
```
Performing report_port on port 'sec_lsb_led[0]'.

****************************************
Report : port
        -verbose
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 17:57:19 2003
****************************************

                          Pin      Wire     Max      Max      Connection
Port            Dir       Load     Load     Trans    Cap      Class       Attrs
-------------------------------------------------------------------------------
min_msb_led[6]  out       0.0110   0.0000   --       --       --
min_msb_led[5]  out       0.0110   0.0000   --       --       --
min_msb_led[4]  out       0.0110   0.0000   --       --       --
min_msb_led[3]  out       0.0110   0.0000   --       --       --
min_msb_led[2]  out       0.0110   0.0000   --       --       --
min_msb_led[1]  out       0.0110   0.0000   --       --       --
min_msb_led[0]  out       0.0110   0.0000   --       --       --
min_lsb_led[6]  out       0.0110   0.0000   --       --       --
min_lsb_led[5]  out       0.0110   0.0000   --       --       --
min_lsb_led[4]  out       0.0110   0.0000   --       --       --
min_lsb_led[3]  out       0.0110   0.0000   --       --       --
min_lsb_led[2]  out       0.0110   0.0000   --       --       --
min_lsb_led[1]  out       0.0110   0.0000   --       --       --
min_lsb_led[0]  out       0.0110   0.0000   --       --       --
sec_msb_led[6]  out       0.0110   0.0000   --       --       --
sec_msb_led[5]  out       0.0110   0.0000   --       --       --
sec_msb_led[4]  out       0.0110   0.0000   --       --       --
sec_msb_led[3]  out       0.0110   0.0000   --       --       --
sec_msb_led[2]  out       0.0110   0.0000   --       --       --

design_analyzer>
```

Command Window

```
                 Max Drive        Min Drive        Resistance       Min      Min      Cell
Input Port       Rise    Fall     Rise    Fall     Max     Min      Cap      Fanout   Deg
-------------------------------------------------------------------------------------------
clk              0.47    0.47     --      --       --      --       --       --       --
cook_time[0]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[1]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[2]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[3]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[4]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[5]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[6]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[7]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[8]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[9]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[10]
                 3.54    3.54     --      --       --      --       --       --       --
cook_time[11]
                 3.54    3.54     --      --       --      --       --       --       --

design_analyzer>
```

Save Constraints & Attributes

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

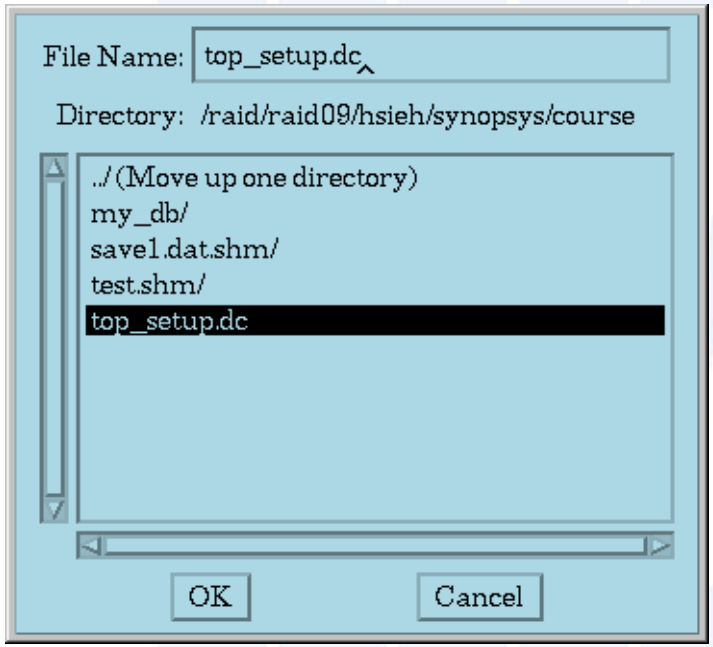


Design Setup File

```
/*****************************************************
  Created by write_script() on Tue Jan 28 10:51:44 2002
*****************************************************/

/* Set the current_design */
current_design top
create_clock -period 50 -waveform {0 25} find(port,"clk")
set_input_delay 10 -max -clock "clk" find(port,"start_cook")
set_output_delay 5.5 -max -clock "clk" find(port,"sec_msb_led[0]")
set_output_delay 5.5 -max -clock "clk" find(port,"sec_msb_led[1]")
set_output_delay 5.5 -max -clock "clk" find(port,"sec_msb_led[2]")
set_output_delay 5.5 -max -clock "clk" find(port,"sec_msb_led[3]")
set_output_delay 5.5 -max -clock "clk" find(port,"sec_msb_led[4]")
set_output_delay 5.5 -max -clock "clk" find(port,"sec_msb_led[5]")
set_output_delay 5.5 -max -clock "clk" find(port,"sec_msb_led[6]")
set_max_delay 5 -from find(port,"start_cook") -to find(port,"sec_msb_led[6]")
set_max_delay 5 -from find(port,"start_cook") -to find(port,"sec_msb_led[5]")
set_max_delay 5 -from find(port,"start_cook") -to find(port,"sec_msb_led[4]")
set_max_delay 5 -from find(port,"start_cook") -to find(port,"sec_msb_led[3]")
……..
```

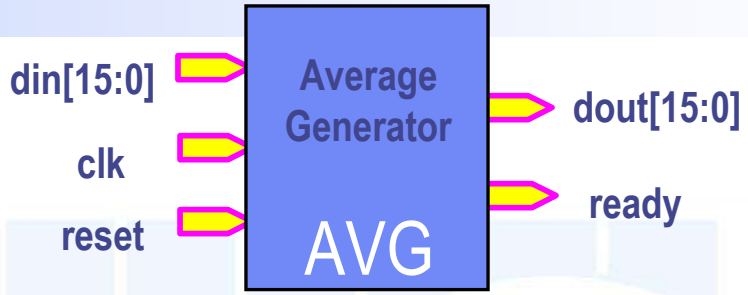
Execute Script File

- Execute dc_shell command script file, use Setup/Execute Script (GUI)



- Or use "`include your_script.scr`" in dc_shell command line

Constraining Example



- Reset the design constraint, set current design and set target library (Initial setup)
- Read the design files
- The period of *clk* is 10ns, rise at time=0, duty cycle of 30%, active high
- For *din[15:0],* the minimum input delay is 2ns and the maximum input delay is 7ns.
- *reset* arrives 3.5 ns after the clock
- For *all outputs*, output delay is 2ns
- *All inputs* except *clk* are driven by "Z" pin of buffd7 from cb35os142 cell library.
- *All outputs* are loaded by 3 times the load of the I pin of buffd1 from the cb35os142

- Avg design contains a multicycle path (3 cycles) from I1/Q to I2/A
- Avg design contains a false path from I1/A through I2/B to I3/C
- Use "8000" wireload model and the wireload mode is enclosed
- Use the NCCOM operation conditions
- Set the max transition time=3 for *all outputs*
- Set the max max_capacitance = 5 for *all outputs*
- Set the fanout load =3 for *all outputs*
- Set the max fanout load = 10 for *all inputs*
- check_timing and check netlist
- Compile the design
- Write area and timing report
- Write HDL netlist as avg.vg
- Write Stand Delay Format (SDF) file (avg.sdf)

Constraining Solution (Synopsys)

- reset_design –design
- read -format verilog {"avg.v"}
- current_design AVG
- create_clock –period 10 –w {0,3} –n clk \
  find (port, "clk")
- set_input_delay –max 7 –clock clk \
  find (port, "din")
- set_input_delay –min 2 –clock clk \
  find (port, "din")
- set_input_delay 3.5 –clock clk \
  find (port, "reset");
- set_output_delay 2 –clock clk all_outputs()
- set_drive_cell –lib cb35os142 –cell buffd7 \
  –pin Z all_inputs() –find(port,"clk)
- set_load load_of (cb35os142/buffd1/I)*3 \
  all_outputs()

- set_multicycle_path 3 –from I1/Q –to I2/A2
- set_false_path –from {I1/A} –through {I2/B} –to {I3/C}
- set_wire_load "8000" -mode enclosed
- set_operating_conditions "NCCOM"
- set_max_transition 3 all_outputs()
- set_max_capacitance 5 all_outputs()
- set_fanout_load 3 all_outputs()
- set_max_fanout 10 all_inputs()
- check_design > check.design
- check_timing > check.timing
- compile -map_effort medium
- report_area report.area
- report_timing report.timing
- write -format verilog -hierarchy –output avg.vg
- write_sdf -version 1.0 -context verilog avg.sdf

Summary (1/2)

- Setting the Real Design Environment
  - Input delay, output delay
  - Input drive strength, output loading
  - Operating condition
  - Wireload model
- Setting the Design Rule constraints (DRC)
  - Maximum fanout
  - Maximum transition time
  - Maximum capacitance

Summary (2/2)

- Setting design constraint
  - Maximum delay, minimum delay
  - Specify clock
  - Maximum area
  - False path, multi-cycle path and multi-frequency
- Handle multiple design instance
  - dont_touch
  - ungroup
  - uniquify
- Check your work before compiling the design
- Save setup file & execute script file

Design Optimization

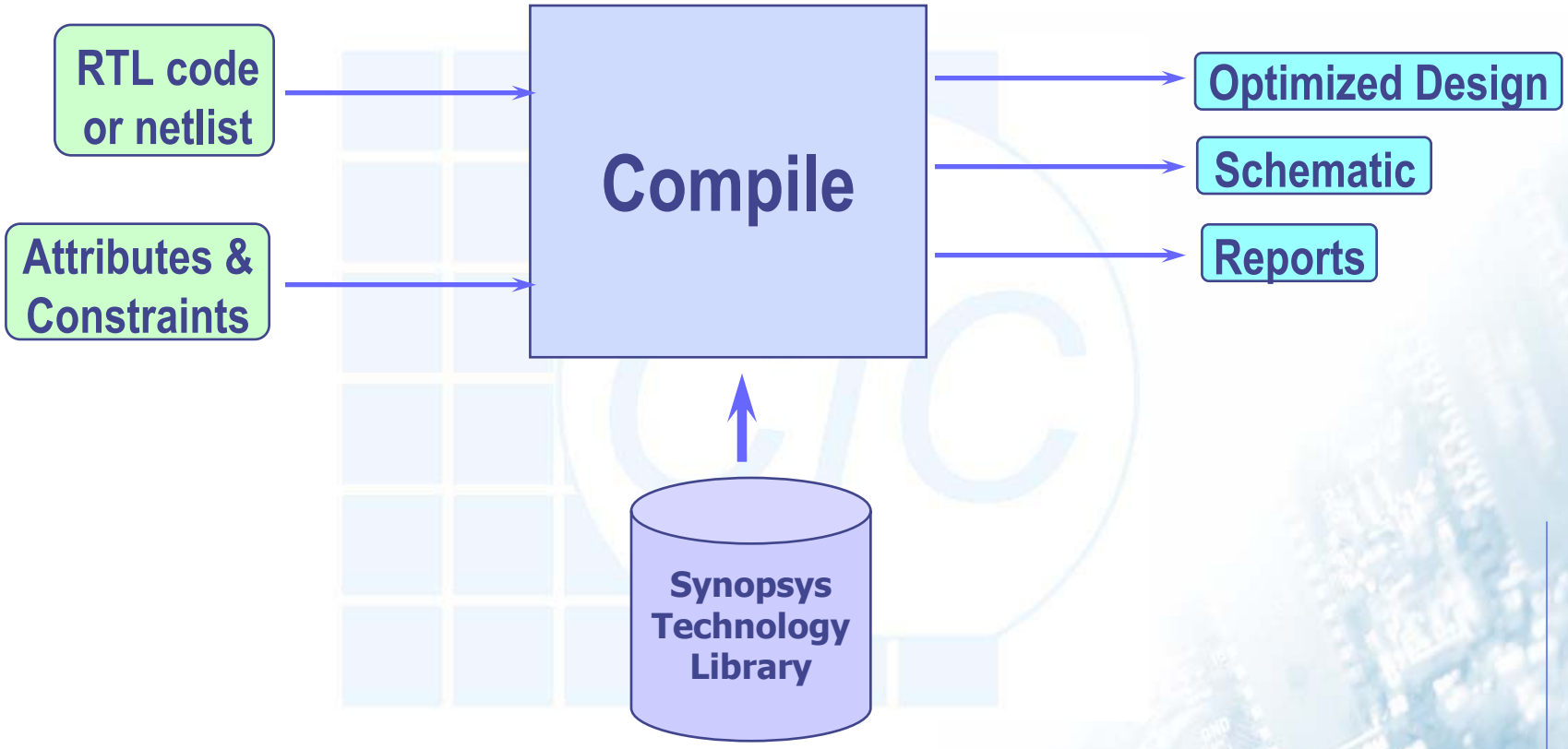
- Design Optimization
  - Compile the Design
  - Finite State Machine Optimization

Compile the Design

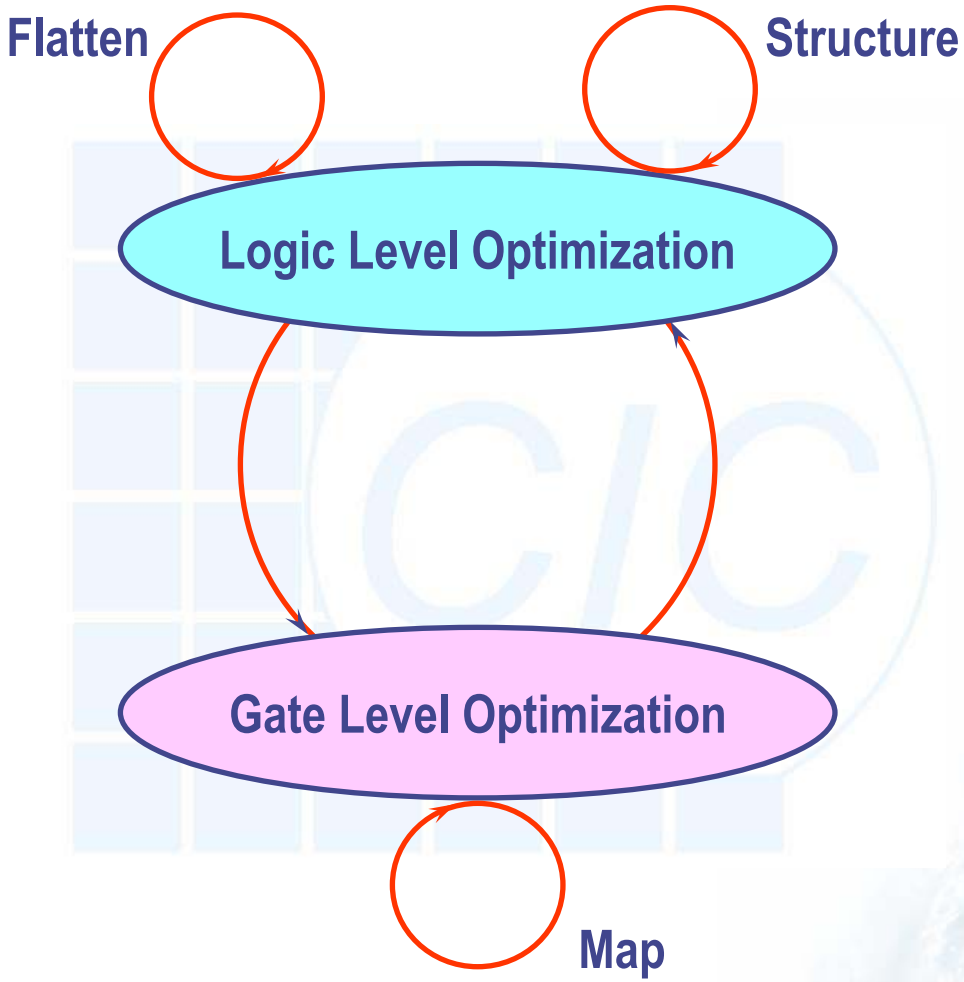
Compile: the "art" of Synthesis (1/3)

- `compile` command is design optimization
- Logic level Optimization
  - flatten (off by default) : removes structure
  - structure : minimizes generic logic
- Gate level Optimization
  - map : makes design technology dependent

Compile (2/3)


RTL code or netlist
Attributes & Constraints
Compile
Optimized Design
Schematic
Reports
Synopsys Technology Library

Compile (3/3)


Flatten
Structure
Logic Level Optimization
Gate Level Optimization
Map

Logic Level Optimization

- Operate with Boolean representation of a circuit
- Has a global effect on the overall area/speed characteristic of a design
- Strategy
  - structure
  - Flatten
  - If both are true, the design is first flattened and then structured

Structure

- Factors out common sub-expression as intermediate variable
- Useful for speed optimization as well as area optimization
- The default logic-level optimization strategy; suitable for structured circuits (e.g. adders and ALU's)
- Example:

Before Structuring

```
f = acd + bcd + e
g = ae' + be'
h = cde
```

After Structuring

```
f = xy + e
g = xe'
h = ye
x = a + b
y = cd
```

Structure Options

- Two options can be used:
  - `set_structure -timing flag`
    - flag can be true or false
    - true (by default) considers timing constraints during structuring
  - `set_structure –boolean flag`
    - flag can be true or false
    - Default is "false", true indicates to use boolean algebra to reduce size of a design.
      (a)(~a) = 0, (a) + (~a) =1, (a) + (a) = a

Flatten

- Flatten is default OFF
- Remove all intermediate variable
- Result a two-level sum-of-product form
  - Note: it doesn't mean that you will have a 2-level hardware due to library limitations
- Flatten is default OFF ;Use when you have a timing goal and have don't cares(x) in your HDL code.
- Example:

Before Flattening

```
f0 = at
f1 = d + t
f2 = t'e
t = b + c
```

After Flattening

```
f0 = ab + ac
f1 = b + c + e
f2 = b'c'e
```

Flatten Options

- Three options can be used:
  - `set_flatten -effort effort`
    - effort can be low, medium or high
  - `set_flatten –minimize  method`
    - method can be none, single, or multiple
    - Single: Single output minimization works on each output , it's generally faster but increases area and loading on inputs
    - Multiple: Multiple output shares product terms between outputs. It's generally more area efficient
  - `set_flatten –phase  flag`
    - flag can be true or false
    - true indicates flatten will also evaluates the phase solution of a karnaugh map.

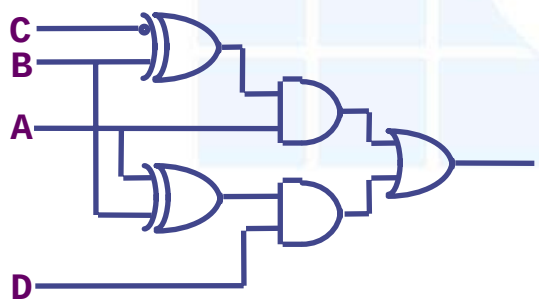
Structuring & Flattening (review)

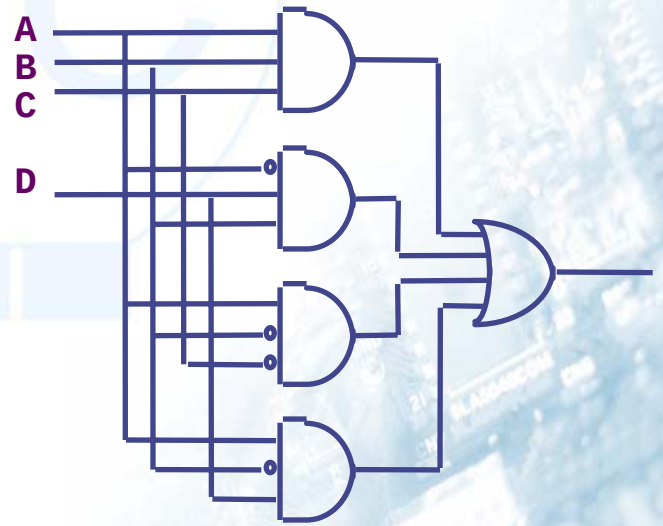
Structuring

- Structure is default ON
- Factors out common subexpression as intermediate variable
- Can help both area and speed of a design
- Suitable for structured circuits

Flattening

- Flatten is default OFF
- Remove all intermediate variable
- Result a two-level sum-of-product form
- Suitable for unstructured circuits





Apply Structure & Flatten

- Attributes/Optimization Directives/Design

Design Name: /raid/raid09/hsieh/synopsys/course/top.db:top

☐ Ungroup ☐ Don't Touch

☐ Boundary Optimization ☐ Disable Wired Logic (ECL)

Sequential Elements: Flip Flop... Latch...

☐ Port is Pad Design Pad Attributes...

Test Scan Style: None

☐ Flatten Logic

Flatten Effort: Flatten Minimize: Flatten Phase:

◆ Low ◆ Single Output ◆ Don't Apply

◇ Medium ◇ Multiple Output ◇ Apply Strategy

◇ High ◇ None

■ Structure Logic

■ Apply Timing Driven Structuring

☐ Apply Boolean Optimization

Apply Cancel

Gate Level Optimization

- Select components to meet timing, design rule & area goals specified for the circuit
- Has a local effect on the area/speed characteristics of a design
- Strategy
  - Mapping
    - Combination mapping
    - Sequential Mapping

Combinational vs. Sequential Mapping

Combinational Mapping

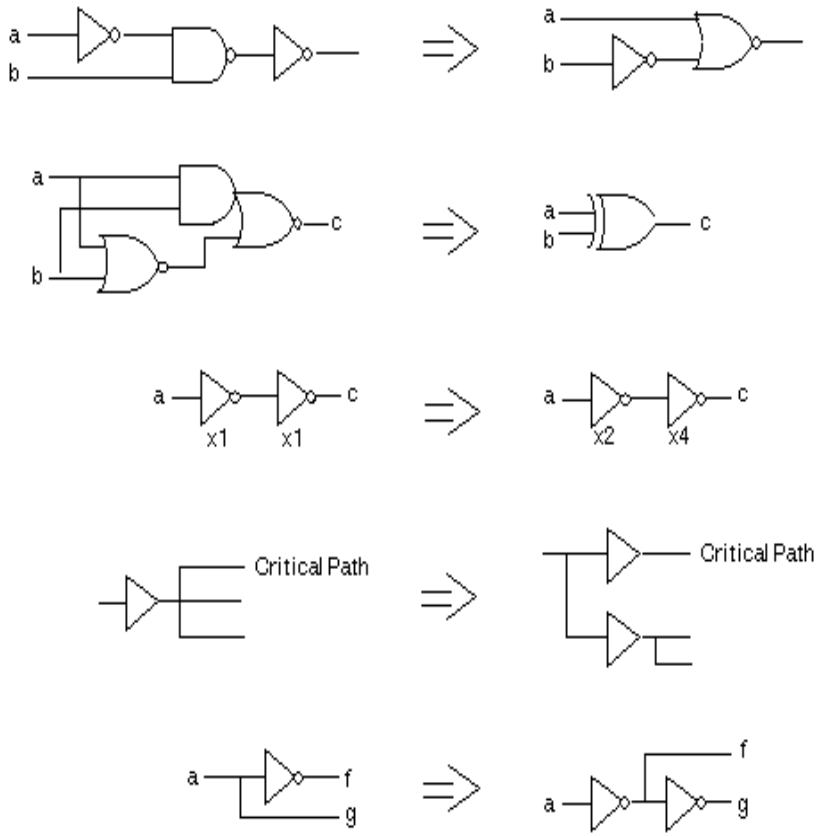
- Mapping rearranges components, combining and re-combining logic into different components
- May use different algorithms such as cloning, resizing or buffering
- Try to meet the design rule constraints and and timing/area goals

Sequential Mapping

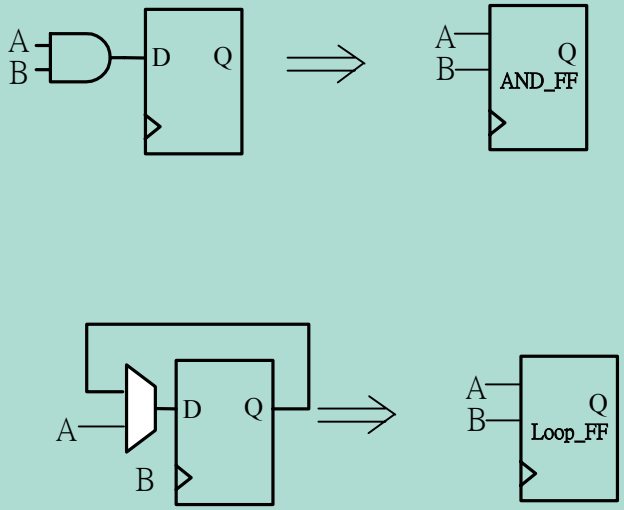
- Optimize the mapping to sequential cells from technology library
- Analyze combinational surrounding a sequential cell to see if it can absorb the logic attribute with HDL
- Try to save speed and area by using a more complex sequential cell

Mapping (cont.)

Combinational mapping

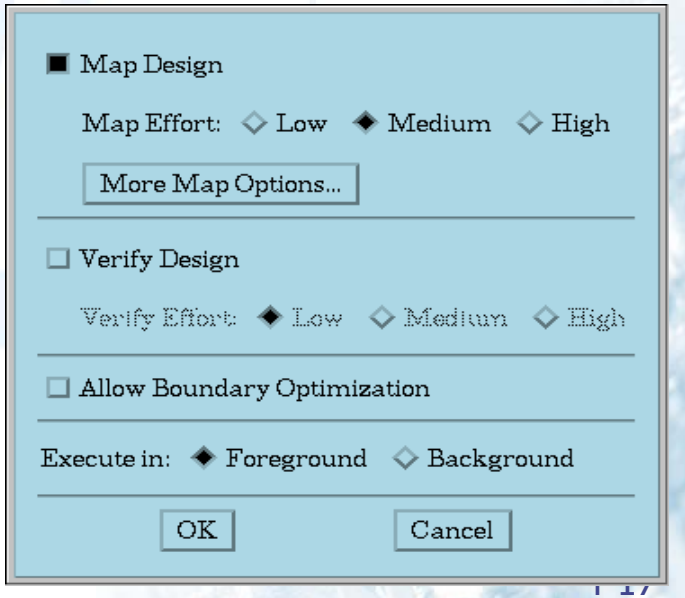


Sequential mapping



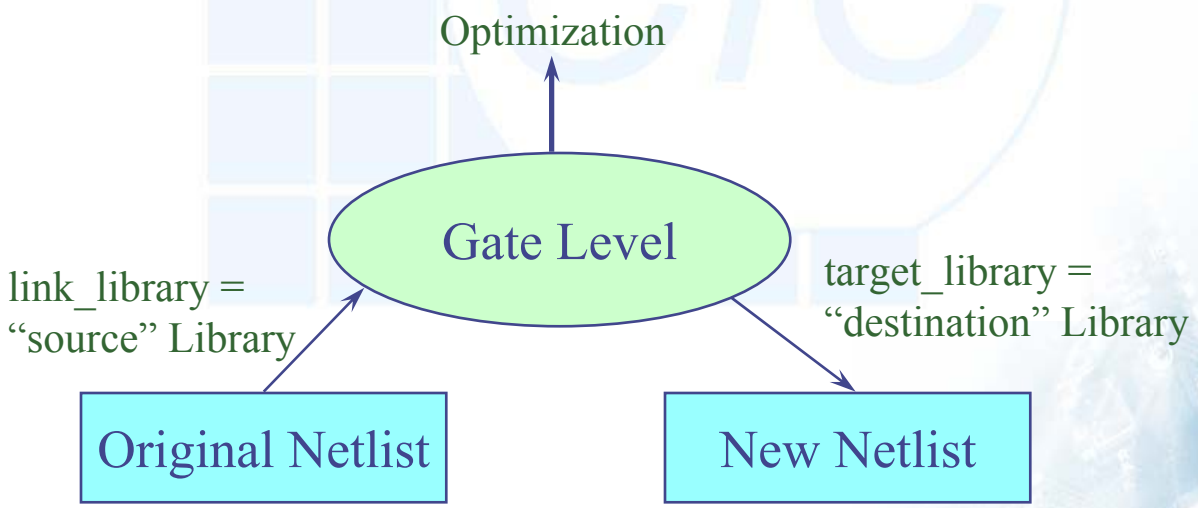
Mapping Effort

- Three effort levels, low, medium, high, determine relative amount of CPU time spent during mapping phase of compile
  - low - quicker synthesis, does not do all algorithms
  - medium - default, good for many design
  - high - it does critical path re-synthesis; but it will use more CPU time; in some cases the action of compile will not terminate



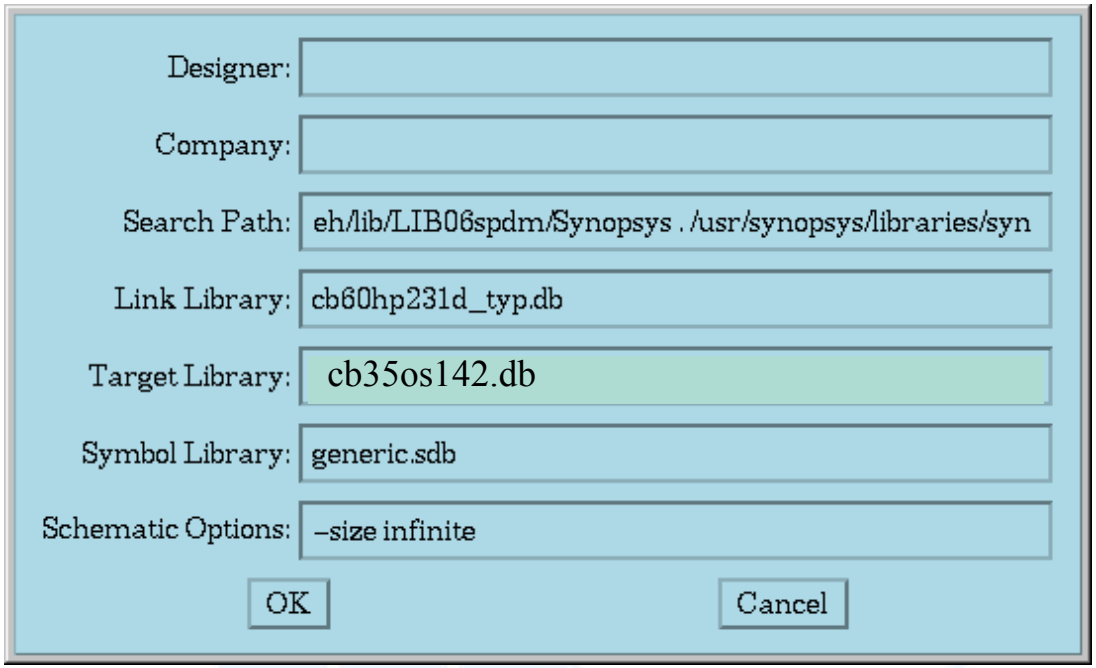
Netlist Translation

- Translate a netlist from one technology library to another
- Design Compiler does a one to one component translation only
- Optimization is not performed
- `dc_shell> translate`



Mapping to Your Target Library

- Setup/Defaults



Note: These environment setting can be initialized using .synopsys_dc.setup file

Start of Optimization

```
design_analyzer> Information: Using 'high' timing effort. (OPT-470)

  Loading target library 'cb60hp230d_typ'
  Loading design 'top'
  Processing FSMs

  Beginning CMOS optimization
  ---------------------------

  Beginning Resource Allocation  (constraint driven)
  ----------------------------
  Structuring 'micro_st'
  Mapping 'micro_st'
  Structuring 'timer'
  Mapping 'timer'
  Structuring 'converter_0'
  Mapping 'converter_0'
  Structuring 'converter_1'
  Mapping 'converter_1'
  Structuring 'converter_2'
  Mapping 'converter_2'
  Structuring 'converter_3'
  Mapping 'converter_3'
  Structuring 'loader'
```

Show | Next | Previous | Cancel

Mapping Optimization

```
Beginning Mapping Optimizations  (Medium effort)
----------------------------------
Structuring 'micro_st'
Mapping 'micro_st'
Structuring 'timer'
Mapping 'timer'
Structuring 'converter_0'
Mapping 'converter_0'
Structuring 'converter_1'
Mapping 'converter_1'
Structuring 'converter_2'
Mapping 'converter_2'
Structuring 'converter_3'
Mapping 'converter_3'
Structuring 'loader'
Mapping 'loader'

        TRIALS    AREA    DELTA DELAY   OPTIMIZATION   DESIGN RULE
                                            COST           COST
        --------  ------  ----------    ------         ---------
           185    585.0     1.23          1.2            369.2
           110    585.0     1.22          1.2            369.2
             8    585.0     1.21          1.2            369.1
```

Show | Next | Previous | Cancel

Number of rules tried

Design area

Worst case delta delay

DRC and optimization Cost function

Cost Function

- Constraint difference: $\triangle = actual - target$
- Design Compiler attempts to reduce the cost functions to 0
- Design Rule Constraint cost function
  - $Cost_{DRC} = \sum\triangle_{max\_fanout} + \sum\triangle_{max\_transition} + \sum\triangle_{max\_cap}$
- Optimization Constraint cost function
  - $Cost_{optimization} = \sum\triangle_{max\_delay} + \sum\triangle_{min\_delay} + \sum\triangle_{max\_area} + \sum\triangle_{max\_power}$

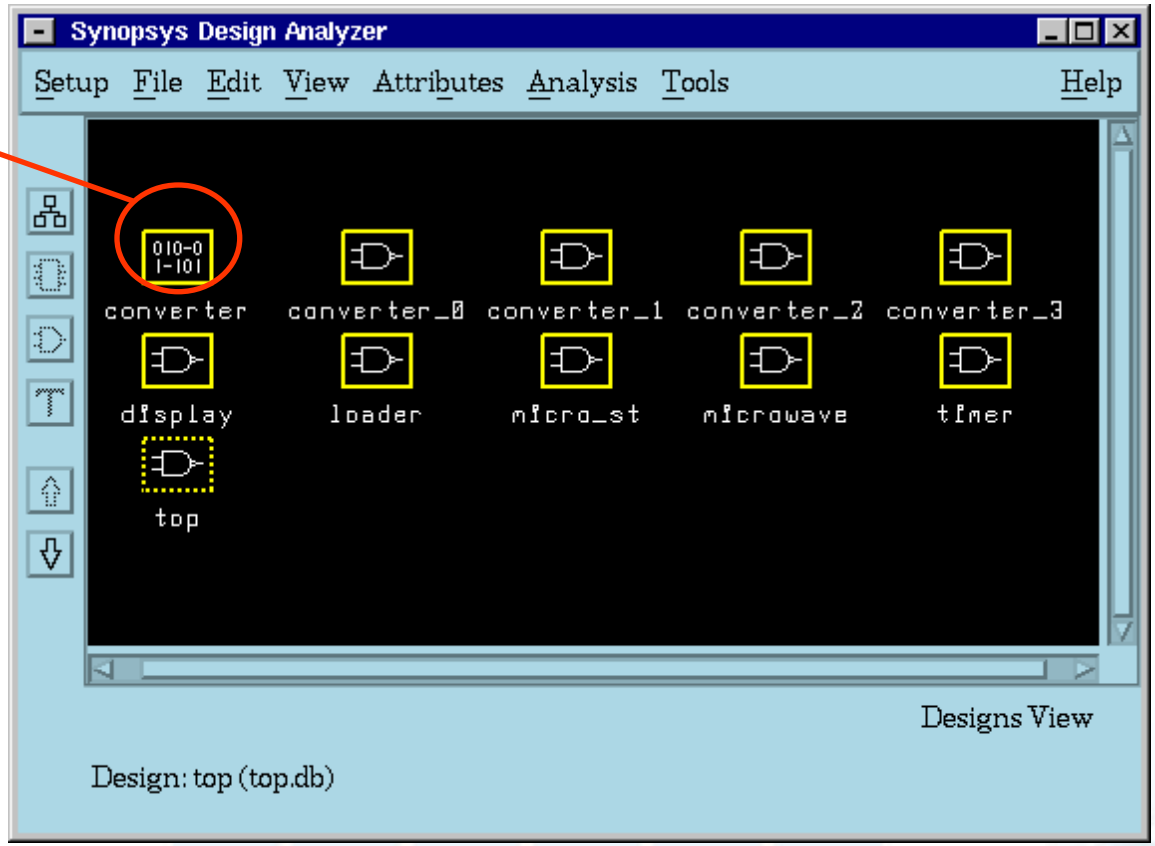
Optimization Conclusion

```
                5      641.6          0.00          0.0           0.1
                6      642.4          0.00          0.0           0.0
                1      641.9          0.00          0.0           0.0
                1
          --------
              638


 Optimization complete
 ---------------------
  Transferring Design 'timer' to database 'timer.db'
  Transferring Design 'converter_0' to database 'top.db'
  Transferring Design 'converter_1' to database 'top.db'
  Transferring Design 'converter_2' to database 'top.db'
  Transferring Design 'converter_3' to database 'top.db'
  Transferring Design 'display' to database 'display.db'
  Transferring Design 'loader' to database 'loader.db'
  Transferring Design 'micro_st' to database 'micro_st.db'
  Transferring Design 'microwave' to database 'microwave.db'
  Transferring Design 'top' to database 'top.db'
Current design is 'top'.
1
design_analyzer> 1
design_analyzer>
```

Show | Next | Previous | Cancel

After Compile

Why ?



Compile Command Options

- Compile options for fine tuning your design:
  - compile -map_effort effort
    - effort ➔ high, medium, low
  - compile -incremental_mapping
  - compile -prioritize_min_path -only_design_rule
  - compile -boundary_optimization

Compile - map_effort & -incremental_mapping

- Compile design with default effort level (medium) for best results
  - `compile`
- Compile design with map effort high
  - it does critical path re-synthesis; but it will use more CPU time; in some cases the action of compile will not terminate
  - `compile -map_effort high`
- For a preliminary estimate of area use map effort low
  - `compile -map_effort low`
- Perform incremental gate level optimization but no logic level optimization
  - `compile -incremental_mapping –map_effort high`

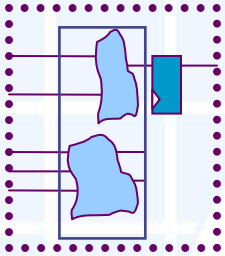
Compile - Min Path Violations

- If you meet the problem as follows
  - My design works fine for worst case conditions but when I check it for best case, I have minimum delay violations!
  - How to solve this problem
    - `compile -prioritize_min_paths -only_design_rule`
    - `-prioritize_min_paths` treats min_delay & fix_hold as Design Rule Violations
    - `-only_design_rule` takes less time than regular compile because it is incremental, i.e., only gate level optimization (mapping) takes place, no logic level optimization takes place

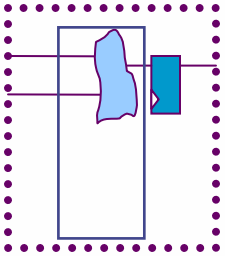
Compile – Boundary Optimization

- DC can do some optimization across boundaries
  1. Removes logic driving unconnected output ports

compile

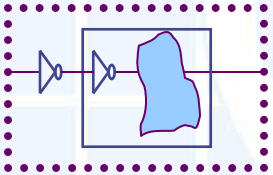


compile –boundary_optimization

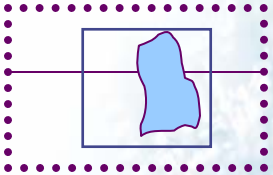


  2. Removes redundant inverters across boundaries

compile

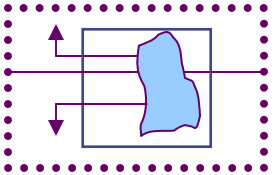


compile –boundary_optimization

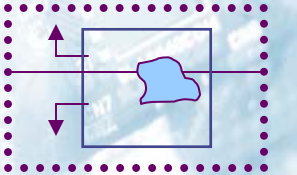


  3. Propagates constants to reduce logic

compile



compile –boundary_optimization

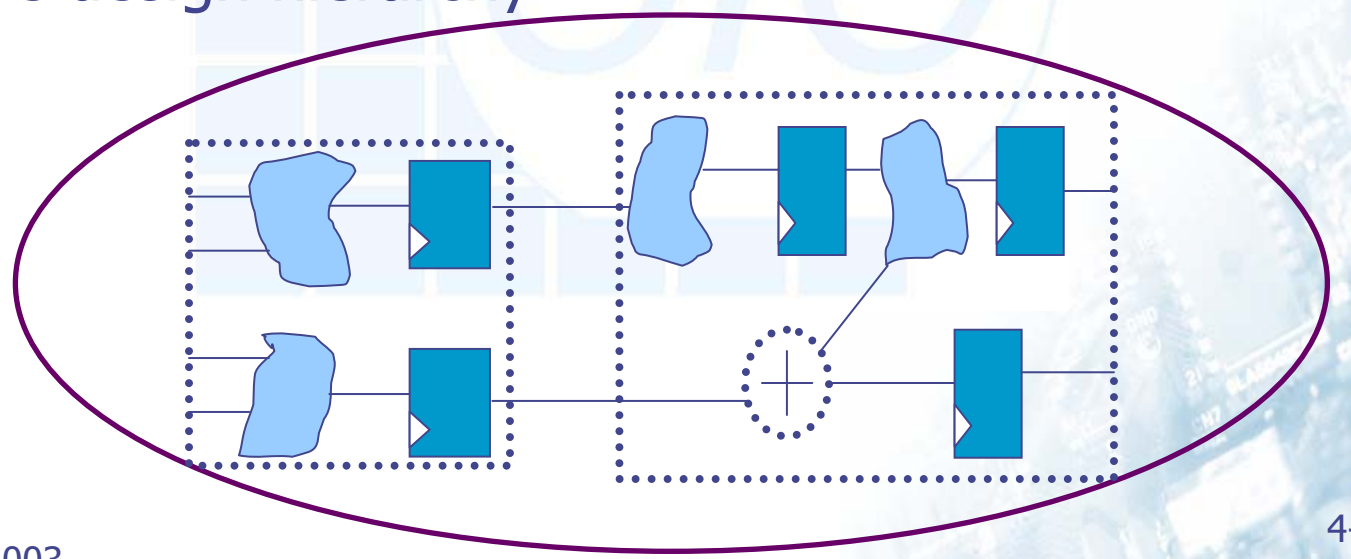


Hierarchical compilation techniques

- Two Strategies for compiling a large hierarchical design
  - Top-down hierarchical compile
  - Bottom-up hierarchical compile

Top-Down Strategy (1/2)

- Read in the entire design hierarchy
- Resolve multiple design instance problem
  (uniquify, compile+dont_touch, ungroup)
- Apply constraints & attributes at the top level design
- Compile from the top level
- Assess results
- Save design hierarchy



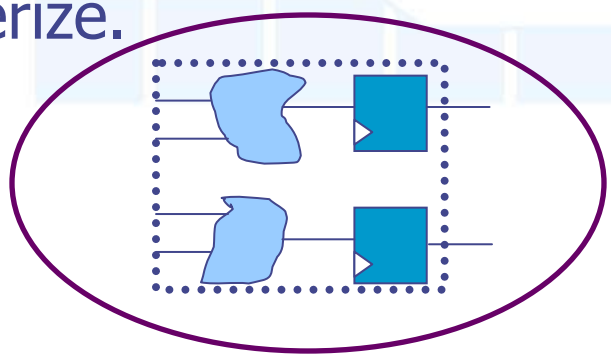
Top-Down Strategy (2/2)

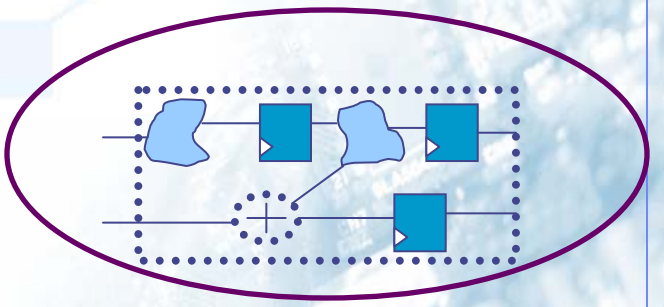
- Advantage:
  1. Inter-module dependencies are taken care of automatically
  2. Less time spent to drive the tool.

- Disadvantage:
  1. Long compile run time with large designs or complex constraints
  2. Even for designs without aggressive timing constraints, can be very memory and CPU intensive

Bottom Up Strategy (2/2)

- Each module is individually synthesized by using estimates for drive, load, input/output delay etc..
- Make sure each module meet their initial constraints
- Read in the entire compiled design and apply the top-level constraints
- Check if the constraint is met, if design passes, you are done!
- If not, pick the worst cell and characterize it.
- Use `write_script` to save the information from characterize.





Bottom Up Strategy (2/2)

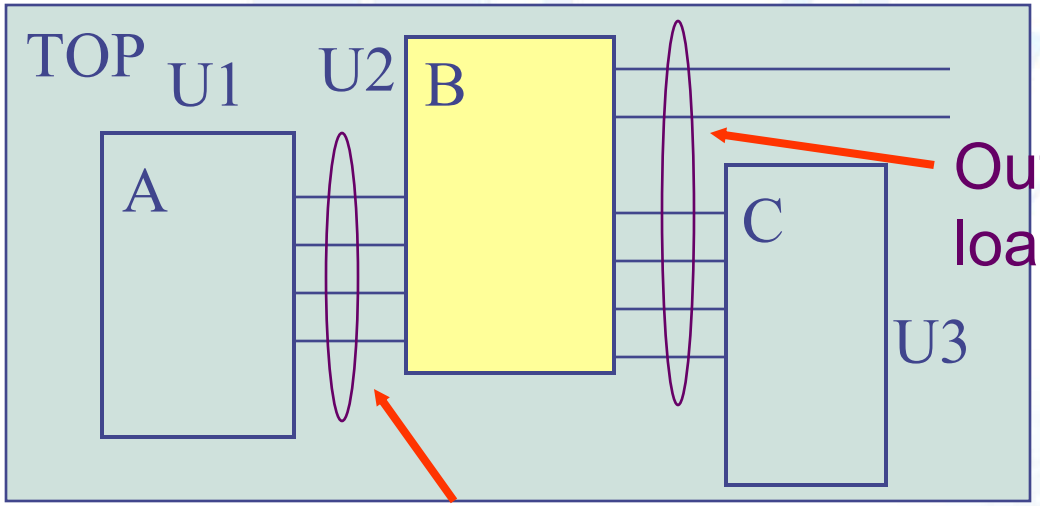
- Advantages:
  1. Not limited by available memory.
  2. Allow for time budgeting

- Disadvantage:
  1. Require iterations until the interfaces of blocks are stable
  2. Require careful revision control

Characterize

- Calculate the actual attributes and top-level constraints imposed on a cell by its surrounding.
- Place those constraints on the *cell*
- Example:

```
characterize  -constraints  find(cell, U2)
```



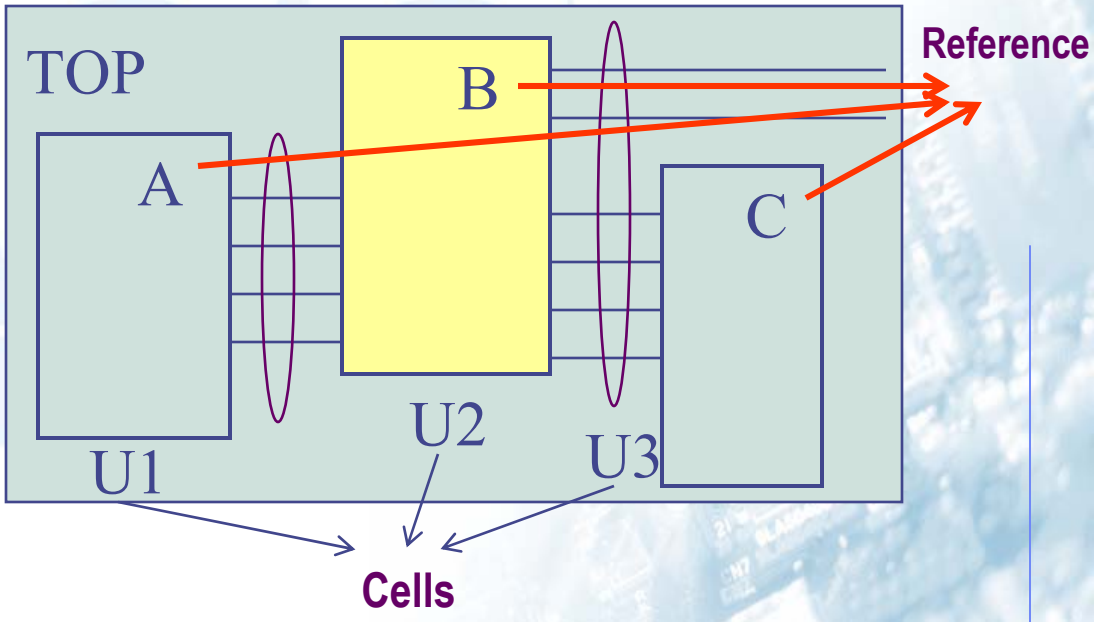
Output delay and loading

Input delay and drive on inputs

Characterize and write_script

- Characterize operates on a cell, but the result is applied to the referenced design
- `write_script` saves the attributes and constraints of current_design

```
current_design top
characterize -constraints
find (cell "U2")
current_design B
write_script > B.scr
(include B.scr)
compile
```



Characterize Limitation

- Can only be used when all blocks are compiled
- Can not be used to derive design budgets
  - All blocks must already be constrained and compiled before this command can be used
- Can only be done one block at one time

Compile Summary (1/2)

- Logic level optimization
  - Operate with Boolean representation of a circuit
  - Flattening (off by default)
    - Remove all structure
    - Design with less than 20 inputs, turn structure off (turn flatten on)
  - Structuring
    - Finds common factors to reduce area
    - Timing driven structuring (default)
    - Boolean optimization (area optimization only)

Compile Summary (2/2)

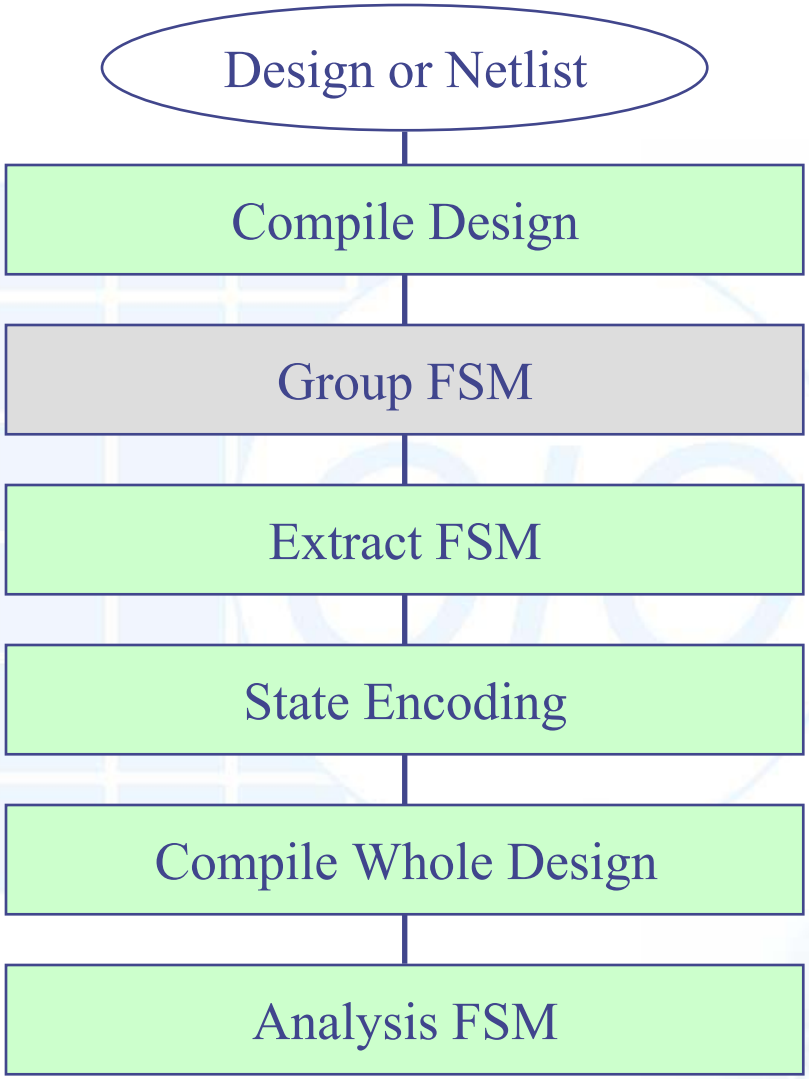
- Gate level optimization
  - Mapping (medium effort is default)
    - Selects components from technology library
    - Use structure from logic level optimization
- Hierarchical compilation techniques
  - top-down
  - bottom up and characterize

Finite State Machine Optimization

Finite State Machine Optimization

- Finite state machine optimization includes
  - State minimization
  - State encoding

FSM Synthesis Design Flow

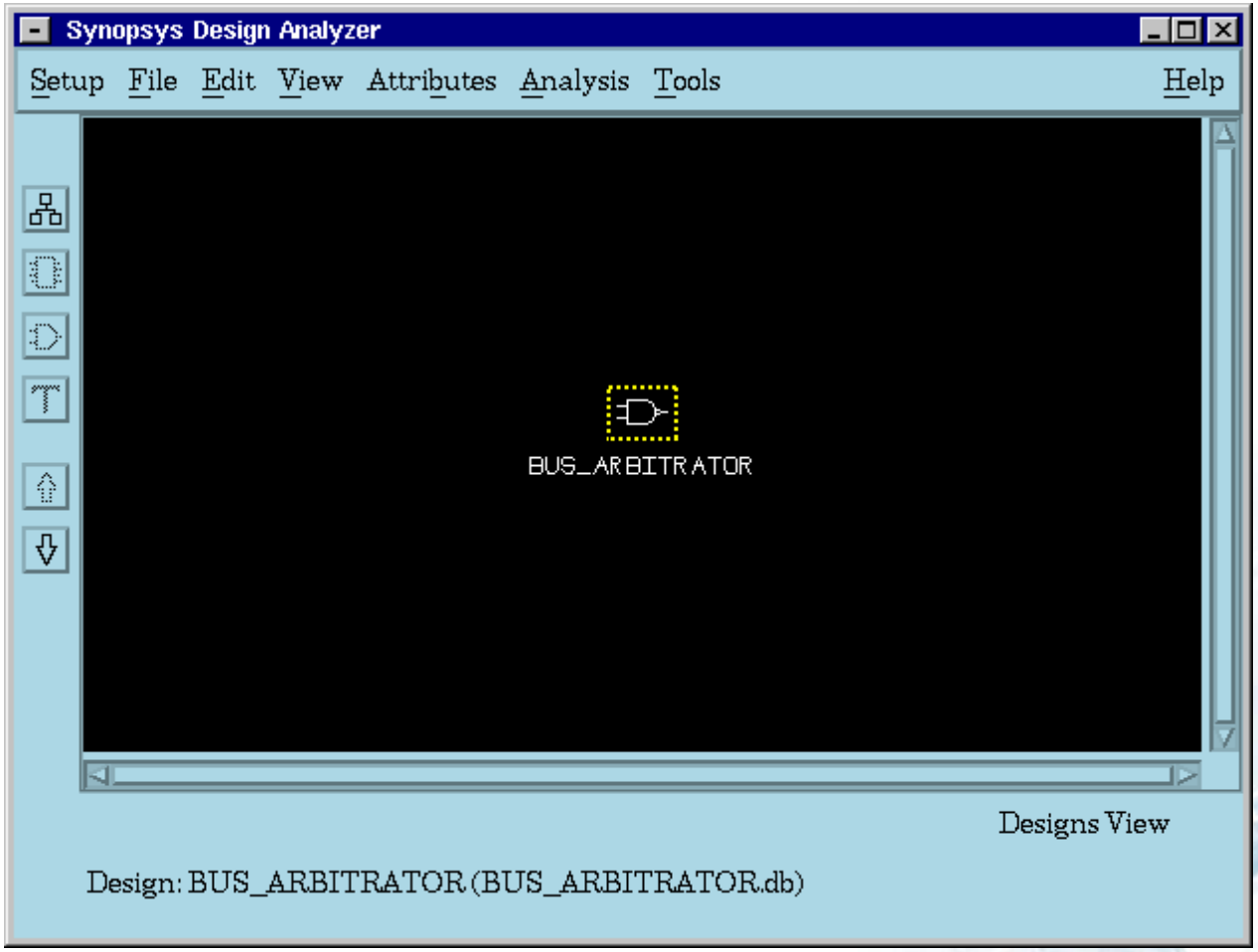


FSM Sample

```
module BUS_ARBITRATOR( REQA, REQB, TIMEUP, CLK, reset,
                       ACKA, ACKB, TIMESTART );

input REQA, REQB, TIMEUP, CLK, reset;
output ACKA, ACKB, TIMESTART;

/* Define states and encodings */
parameter [2:0] // synopsys enum code
          Grant_A=3'b001, Wait_A=3'b011, Timeout_A1=3'b111,
          Grant_B=3'b010, Wait_B=3'b110, Timeout_B1=3'b101;

reg [2:0] /* synopsys enum code */ present_State,
next_State;

//synopsys state_vector present_State

reg ACKA, ACKB, TIMESTART;

 always @ (REQA or REQB or TIMEUP  or reset or
present_State)
        begin
```

Design Flow

- Read design & compile design to gate level

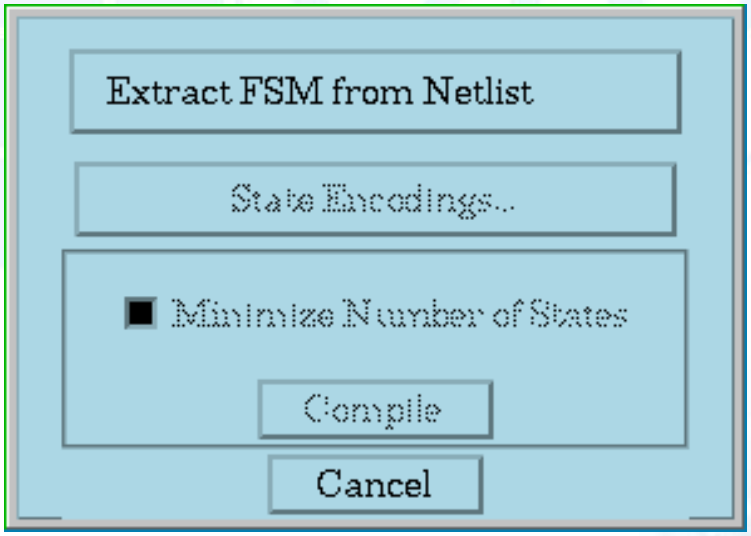


Extract FSM (1/3)

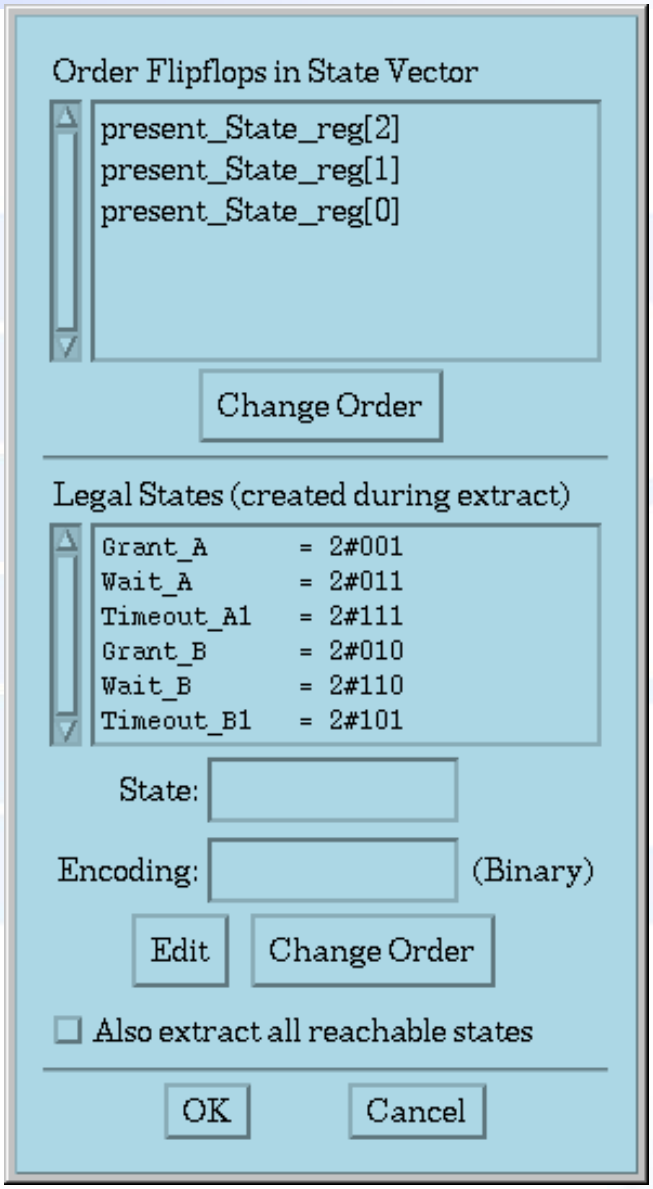
- Generate a state table from your HDL description
- Input Design Requirements
  - Contain only one type of sequential element
  - Does not contain multiple or gated clocks
  - Contain only one reset signal
- If from HDL, use Synopsys directives to keep familiar state name

Extract FSM (2/3)

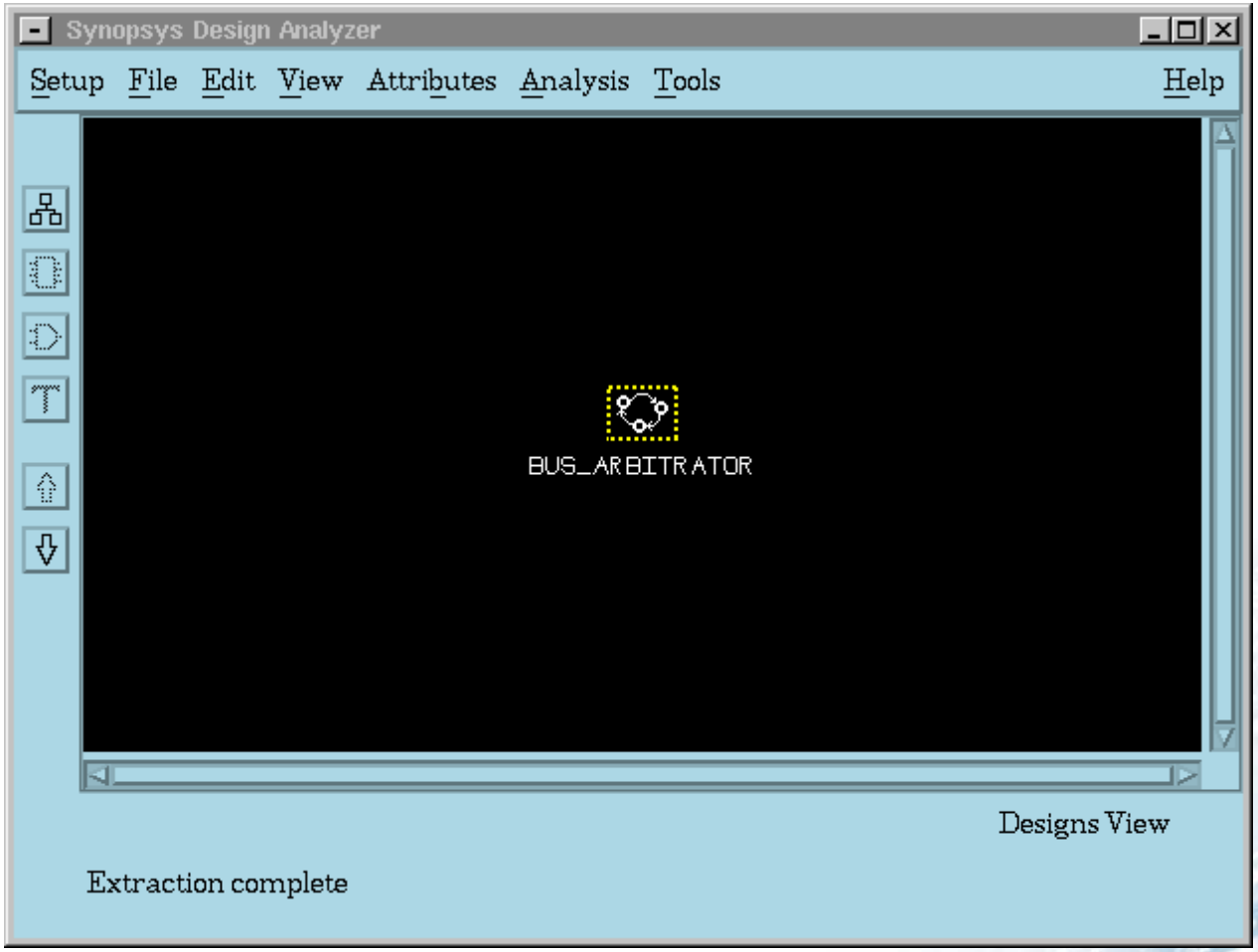
- Design Compiler extracts the state names from HDL
- Resulting state table only has legal states from enumerated directive
- Tools/Finite State Machine
- Extract FSM from Netlist/Define the Legal States



Extract FSM (3/3)



Extracted FSM



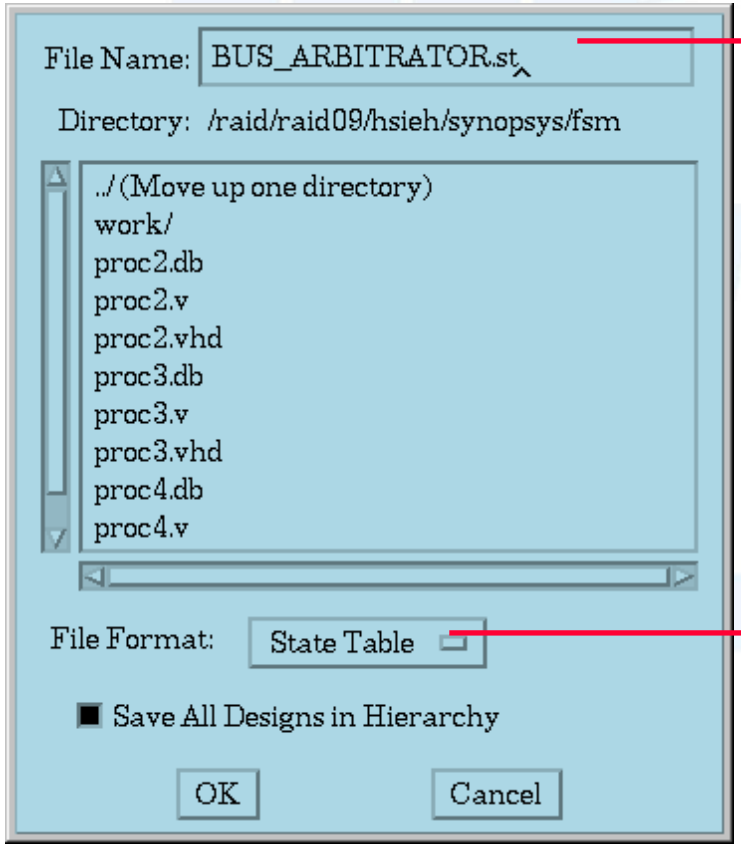
Group FSM

- If there is error message during extracting FSM, like Invalid non_combinational cell "out_reg" you must group FSM and extract FSM again
- To group FSM, use the following dc_shell command

```
dc_shell>set_fsm_state_vector {present_State[2],
                present_State[1], present_State[0]}
dc_shell>group -fsm -design_name fsm_name
```

Save as State Table

- You can save your extracted FSM in state table format



set file name extension as .st

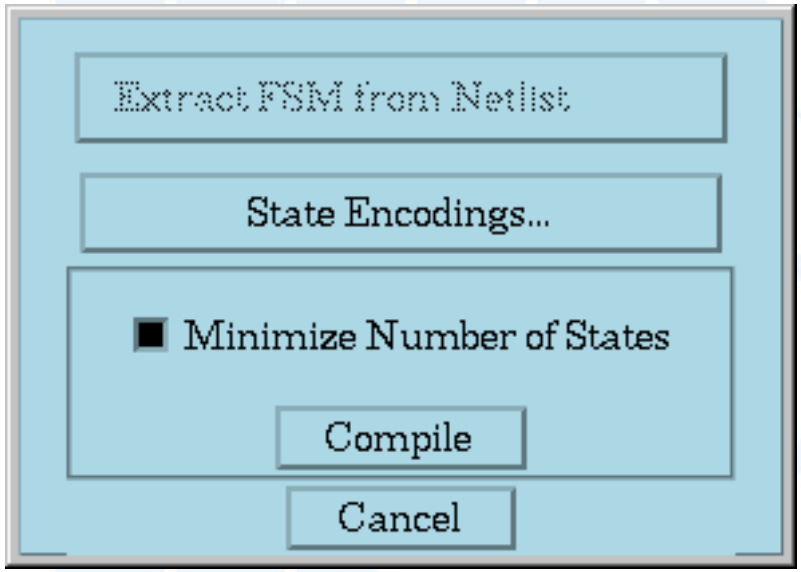
set file format as State Table

State Table File Format

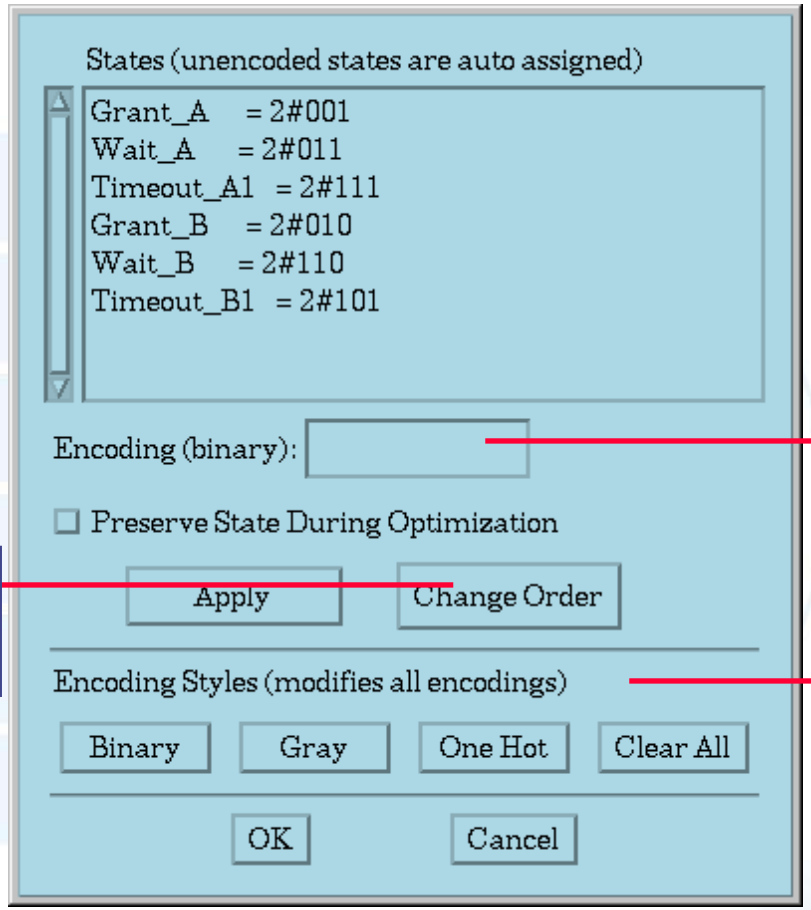
```
.design BUS_ARBITRATOR
.inputnames REQA REQB TIMEUP CLK reset
.outputnames ACKA ACKB TIMESTART
.clock CLK rising_edge

---1 Grant_A    Wait_B     ~~~
0--0 Grant_A    Wait_A     ~~~
10-0 Grant_A    Grant_A    1~1
1110 Grant_A    Timeout_A1 ~~~
1-00 Grant_A    Grant_A    1~1
---- Grant_A    ~          ~0~
0--- Grant_A    ~          0~0
-11- Grant_A    ~          0~0
---1 Grant_A    Wait_B     0~0
0--0 Grant_A    Wait_A     ~~~
10-0 Grant_A    Grant_A    ~~~
1110 Grant_A    Timeout_A1 ~~~
1-00 Grant_A    Grant_A    ~~~
---1 Wait_A     Wait_B     ~~~
1--0 Wait_A     ~          ~~1
-1-0 Wait_A     Grant_B    ~11
```

State Encoding (1/2)

- Tools/Finite State Machines/State Encoding



State Encoding (2/2)

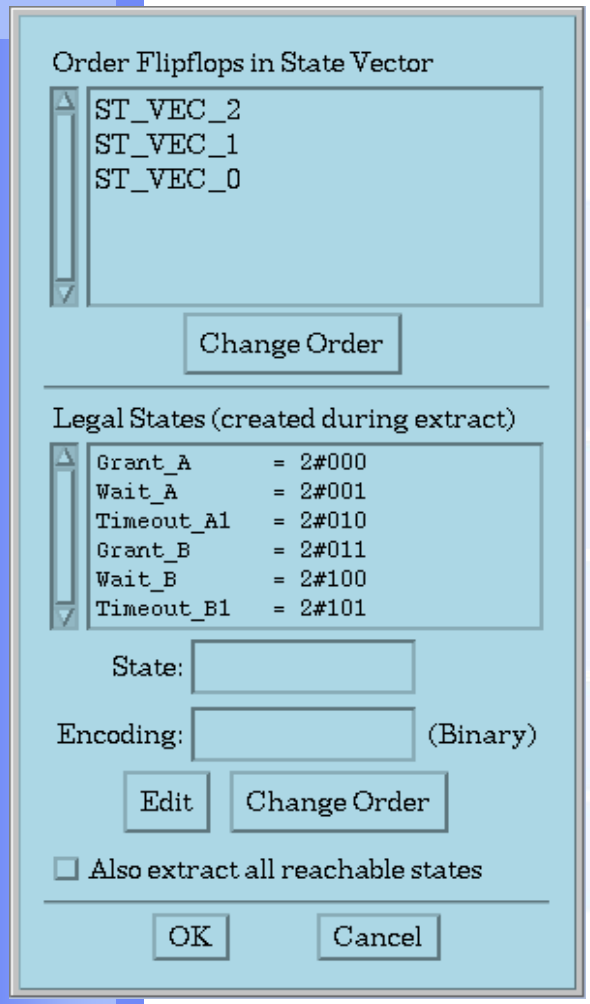


Assign encoding values to states

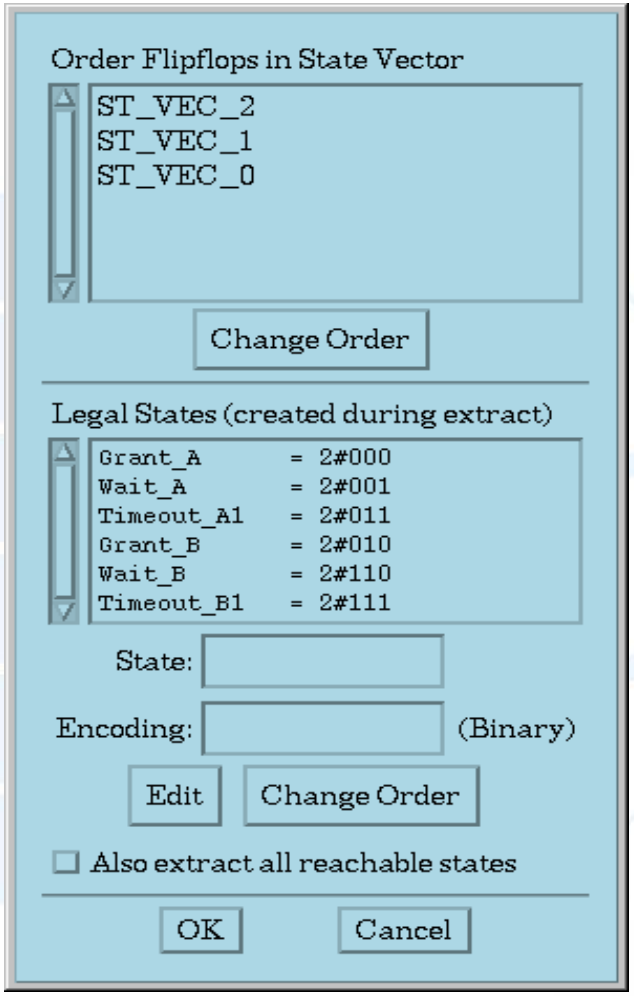
Change states order

Specify encoding style

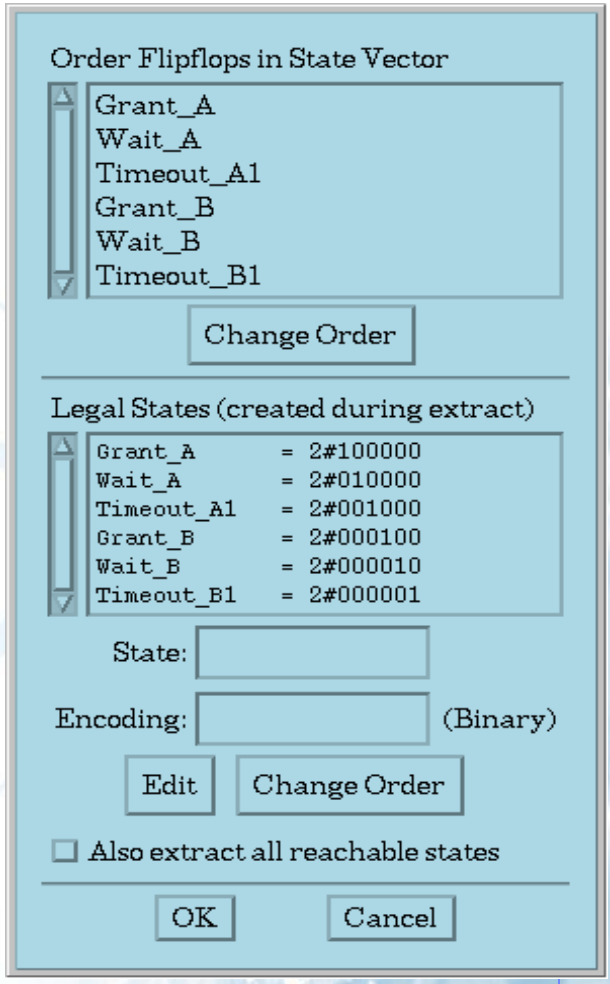
Encoding Style



Binary



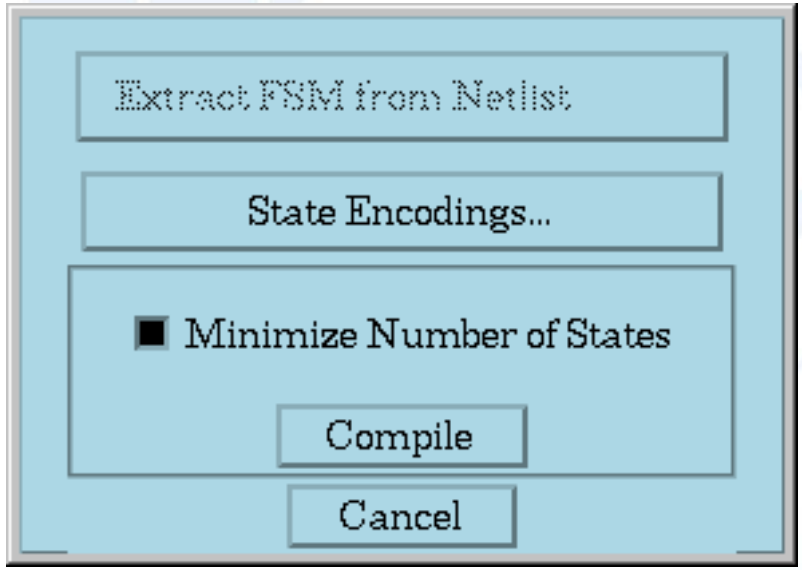
Gray



One-Hot

Compile Design

- Compile the finite state machine alone, or use top down compile methodology to compile the whole hierarchy



Analysis - FSM Report

- FSM report shows FSM attributes & information

Report Output

```
design_analyzer>
****************************************
Report : fsm
Design : micro_st
Version: 2002.05-SP1
Date   : Wed May 14 18:33:45 2003
****************************************

Clock              : clk         Sense: rising_edge
Asynchronous Reset: reset        Sense: falling       State: IDLE

Encoding Bit Length: 3
Encoding style     : auto

State Vector: { state_reg[2] state_reg[1] state_reg[0] }

State Encodings and Order:

IDLE       : 000
LAMP_TEST  : 011
SET_CLOCK  : 001
TIMER      : 010
DONE_MSG   : 110

Merged States: None
```

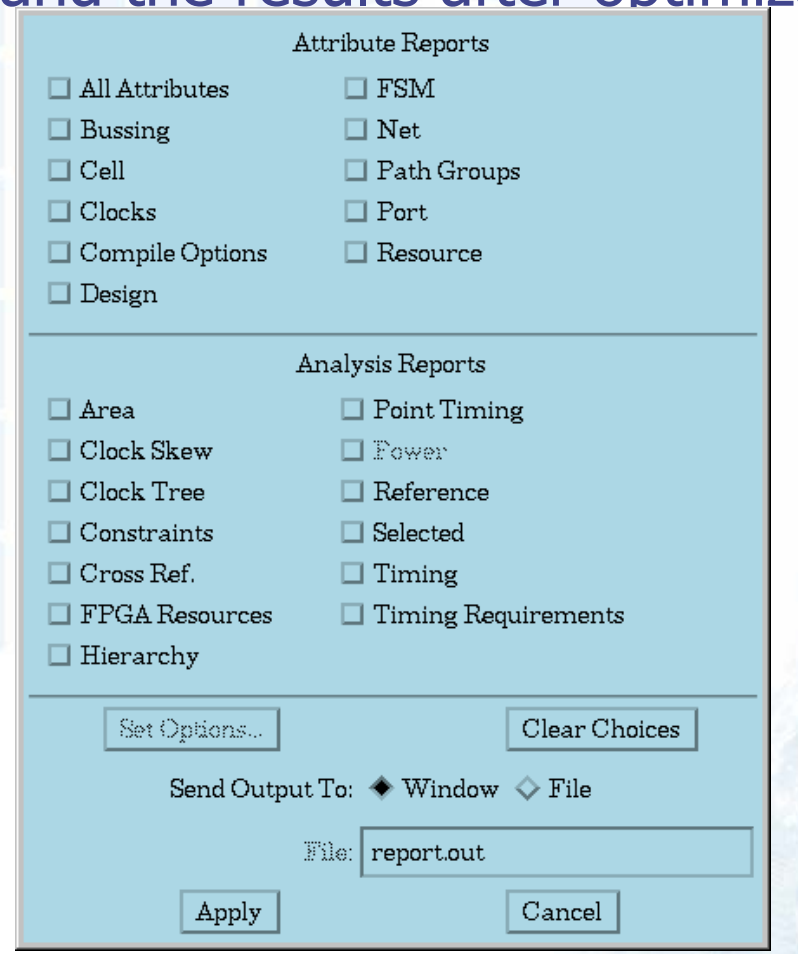
Show | Next | Previous | Cancel

Synthesis Report & Analysis

Report

Analysis/Report

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



Report We will Generate

- Attribute reports
  - All attributes, clock, port, design, net
- Analysis reports
  - Area, hierarchy, constraints, timing, point timing

Net Report

- Net report shows the statical results of each net

Report Output

```
design_analyzer>
****************************************
Report : net
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 18:47:09 2003
****************************************


Operating Conditions: WCCOM   Library: cb35os142
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
top                    ForQA             cb35os142


Attributes:
   dr - drc disabled
    d - dont_touch

Net                 Fanout     Fanin      Load   Resistance    Pins   Attributes
-------------------------------------------------------------------------------
clk                     19         1      0.33         0.00      20   dr, d
cook                     9         1      0.22         0.00      10
cook_time[0]             1         1      0.04         0.00       2
cook_time[1]             1         1      0.04         0.00       2
cook_time[2]             1         1      0.04         0.00       2
cook_time[3]             1         1      0.04         0.00       2
cook_time[4]             1         1      0.04         0.00       2
```

Show Next Previous Cancel

Port Report

- dc_shell command

```
dc_shell> report_port -verbose { port_list }
```

or in the option menu set *verbose*

Command Window

```
Performing report_port on port 'sec_lsb_led[0]'.

****************************************
Report : port
         -verbose
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 17:57:19 2003
****************************************

                          Pin      Wire     Max      Max      Connection
Port            Dir       Load     Load     Trans    Cap      Class       Attrs
-------------------------------------------------------------------------------
min_msb_led[6]  out       0.0110   0.0000   --       --       --
min_msb_led[5]  out       0.0110   0.0000   --       --       --
min_msb_led[4]  out       0.0110   0.0000   --       --       --
min_msb_led[3]  out       0.0110   0.0000   --       --       --
min_msb_led[2]  out       0.0110   0.0000   --       --       --
min_msb_led[1]  out       0.0110   0.0000   --       --       --
min_msb_led[0]  out       0.0110   0.0000   --       --       --
min_lsb_led[6]  out       0.0110   0.0000   --       --       --
min_lsb_led[5]  out       0.0110   0.0000   --       --       --
min_lsb_led[4]  out       0.0110   0.0000   --       --       --
min_lsb_led[3]  out       0.0110   0.0000   --       --       --
min_lsb_led[2]  out       0.0110   0.0000   --       --       --
min_lsb_led[1]  out       0.0110   0.0000   --       --       --
min_lsb_led[0]  out       0.0110   0.0000   --       --       --
sec_msb_led[6]  out       0.0110   0.0000   --       --       --
sec_msb_led[5]  out       0.0110   0.0000   --       --       --
sec_msb_led[4]  out       0.0110   0.0000   --       --       --
sec_msb_led[3]  out       0.0110   0.0000   --       --       --
sec_msb_led[2]  out       0.0110   0.0000   --       --       --

design_analyzer>
```

Command Window

```
                 Max Drive        Min Drive        Resistance       Min      Min      Cell
Input Port       Rise     Fall    Rise     Fall    Max      Min     Cap      Fanout   Deg
-------------------------------------------------------------------------------------------
clk              0.47     0.47    --       --      --       --      --       --       --
cook_time[0]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[1]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[2]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[3]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[4]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[5]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[6]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[7]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[8]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[9]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[10]
                 3.54     3.54    --       --      --       --      --       --       --
cook_time[11]
                 3.54     3.54    --       --      --       --      --       --       --

design_analyzer>
```

Area Report

- Area report shows the gate count of the design

Report Output

```
design_analyzer> Information: Updating design information... (UID-85)

****************************************
Report : area
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 18:43:47 2003
****************************************

Library(s) Used:
    cb35os142 (File: /cad2/lab/LIB/CIC_CBDK35_V3/Synopsys/cb35os142.db)

Number of ports:                 49
Number of nets:                  84
Number of cells:                  3
Number of references:             3

Combinational area:         418.333344
Noncombinational area       122.666618
Net Interconnect area:        1.340000

Total cell area:            541.000000
Total area:                 542.339966
1
design_analyzer>
```

Show | Next | Previous | Cancel

It means that your design is about 542 gate count

Hierarchy Report

- Hierarchy report shows the component used in your each block & its hierarchy

Report Output

```
design_analyzer>
****************************************
Report : hierarchy
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 18:49:32 2003
****************************************

top
    display
        converter_0
            aoi221d2          cb35os142
            inv0d1            cb35os142
            mi02d1            cb35os142
            nd04d1            cb35os142
            nd12d1            cb35os142
            nd13d1            cb35os142
            oaim21d1          cb35os142
            oaim31d1          cb35os142
            oaim211d1         cb35os142
            or02d1            cb35os142
            xn02d1            cb35os142
        converter_1
            aoi221d2          cb35os142
            inv0d1            cb35os142
            mi02d1            cb35os142
```

Show Next Previous Cancel

Reference Report

- Reference report shows statistical result about references in the design

Report Output

```
****************************************
Report : reference
Design : top
Version: 2002.05-SP1
Date   : Wed May 14 18:51:02 2003
****************************************
.
Attributes:
    b - black box (unknown)
   bo - allows boundary optimization
    d - dont_touch
   mo - map_only
    h - hierarchical
    n - noncombinational
    r - removable
    s - synthetic operator
    u - contains unmapped logic
.
Reference          Library       Unit Area   Count    Total Area   Attributes
-------------------------------------------------------------------------------
display                          170.666656      1    170.666656   h
microwave                         77.333313      1     77.333313   h, n
timer                            293.000000      1    293.000000   h, n
-------------------------------------------------------------------------------
Total 3 references                                    541.000000
1
```

Show | Next | Previous | Cancel

Constraints Report

- Constraints report shows whether compiled design meets your constraints
- `dc_shell > report_constraint  -all_violators`

```
clk                          0.00      1.00      0.00
default                      0.00      1.00      0.00
-----------------------------------------------------
max_delay/setup                                  0.00

                                             Weighted
Group (min_delay/hold)       Cost     Weight     Cost
-----------------------------------------------------
clk (no fix_hold)            0.00      1.00      0.00
default                      0.00      1.00      0.00
-----------------------------------------------------
min_delay/hold                                   0.00

Constraint                                       Cost
-----------------------------------------------------
max_delay/setup                                  0.00 (MET)
max_transition                                   0.00 (MET)
max_fanout                                       0.00 (MET)
max_area                                        41.88 (VIOLATED)

Test Constraint                                  Cost
-----------------------------------------------------
min_fault_coverage                               None  (UNKNOWN)
```

Show Next Previous Cancel

Timing Report (1/2)

- Path information produced by `report_timing` command you should know

Report Output

```
design_analyzer>
****************************************
Report : timing
         -path short
         -delay max
         -max_paths 1
Design : pmult32
Version: 1999.10-5
Date   : Wed Nov 15 09:46:06 2000
****************************************

Operating Conditions: NCCOM   Library: cb35os142
Wire Load Model Mode: enclosed

  Startpoint: A2_reg[5] (rising edge-triggered flip-flop clocked by clock)
  Endpoint: out_reg[55]
            (rising edge-triggered flip-flop clocked by clock)
  Path Group: clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  pmult32            22000                 cb35os142
  pmult32_DW_bthenc_21
```

Show Next Previous Cancel

Timing Report (2/2)

- Path delay time, path required time and summary section.

```
  timer/U135/zn (nr02d0)                                  0.89      40.87 r
  timer/U137/zn (nr02d0)                                  1.31      42.18 f
  timer/U59/z (ni01d1)                                    1.79      43.97 f
  timer/U206/zn (in01d0)                                  1.24      45.21 r
  timer/U139/zn (nr02d0)                                  0.99      46.20 f
  timer/U140/zn (nr03d0)                                  1.35      47.55 r
  timer/U51/zn (in01d0)                                   0.74      48.30 f
  timer/sec_msb_next_reg[3]/d (dfctnb)                    0.00      48.30 f
  data arrival time                                                 48.30

  clock clk (rise edge)                                  50.00      50.00
  clock network delay (ideal)                             0.00      50.00
  timer/sec_msb_next_reg[3]/cp (dfctnb)                   0.00      50.00 r
  library setup time                                     -0.54      49.46
  data required time                                                49.46
  -------------------------------------------------------------------------
  data required time                                                49.46
  data arrival time                                                -48.30
  -------------------------------------------------------------------------
  slack (MET)                                                        1.16


1
design_analyzer>
```

Show | Next | Previous | Cancel

Timing Report

- Timing report shows maximum or minimum delay path of design, the default is to display one maximum delay path

Clock Skew

Net_delay + cell_delay are combined

Unateness

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
timer/min_lsb_next_reg[0]/cp (dfntnb)	0.00	0.00 r
timer/min_lsb_next_reg[0]/q (dfntnb)	2.62	2.62 f
timer/U73/zn (nr02d0)	1.54	4.16 r
timer/U181/zn (nd04d0)	0.00	4.16 f
timer/U182/zn (nr02d0)	1.36	5.52 r
timer/done (timer)	0.00	5.52 r
microwave/done (microwave)	0.00	5.52 r
microwave/U1/z (ni01d1)	2.83	8.35 r
microwave/micro_st/done (micro_st)	0.00	8.35 r
microwave/micro_st/U17/z (an02d1)	1.57	9.92 r
microwave/micro_st/U15/z (or02d1)	1.89	11.81 r
microwave/micro_st/load_done (micro_st)	0.00	11.81 r
microwave/loader/load_done (loader)	0.00	11.81 r
microwave/loader/U27/z (or02d1)	1.83	13.64 r
microwave/loader/U15/z (ni01d3)	2.09	1
microwave/loader/U26/z (or02d1)	2.00	1
microwave/loader/load (loader)	0.00	17.73 r
microwave/load (microwave)	0.00	17.73 r
timer/load (timer)	0.00	17.73 r
timer/U53/z (ni01d3)	2.17	19.89 r

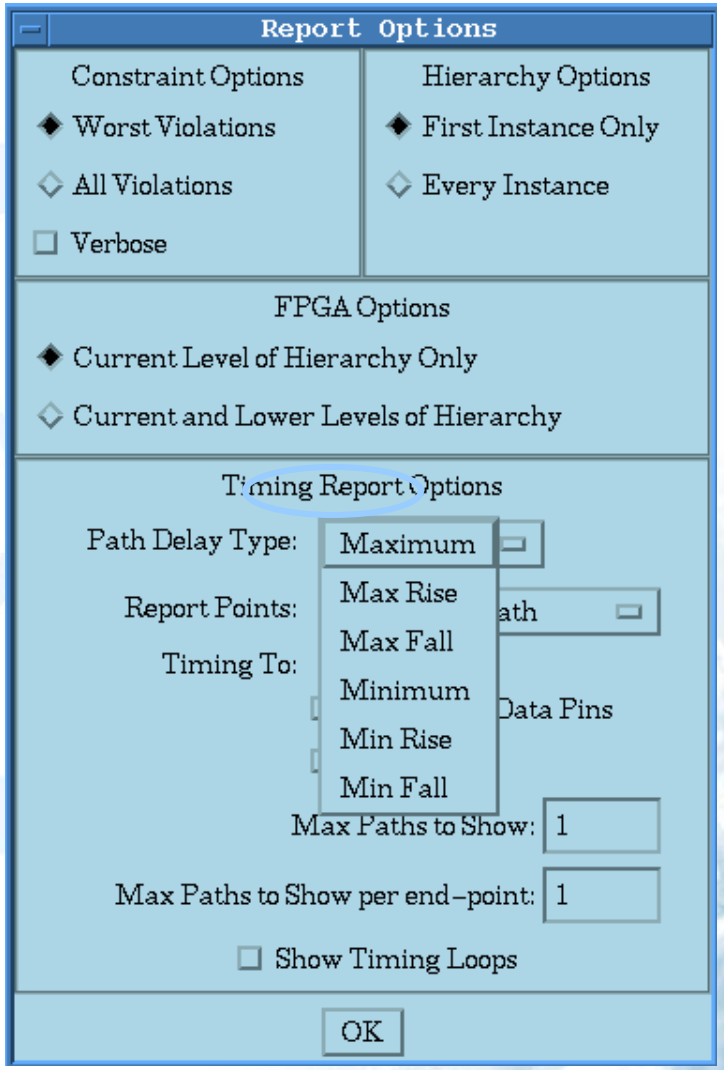
Show Next Previous Cancel

What is Slack ?

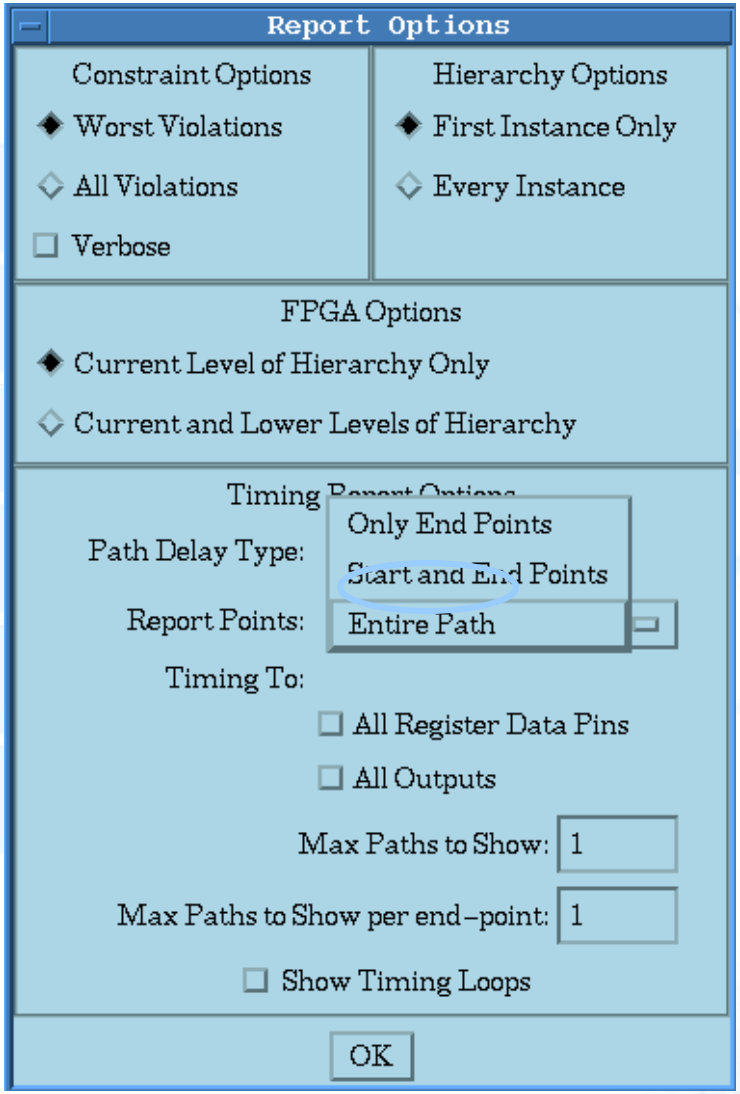
- Slack is the resulting margin between required & actual arrival time
- Positive slack or zero means meet constraints
- Negative slack means violate constraints

Timing Report Options (1/3)

- Modify timing report options for your need



Timing Report Options (2/3)



Timing Report Options (3/3)

- The default is to report the path with the worst slack within each path group
- The default is to report the longest path to output port, if the design has no timing constraint
- Syntax:

```
report_timing
    [-to name_list]
    [-from name_list]
    [-through name_list]
    [-path short | full |end |only]
    [-delay min | max]
    [-input_pins ]
    [-max_paths path count]
    [-nworst path_count]
    [-nets]
```

Timing Report (setup time check)

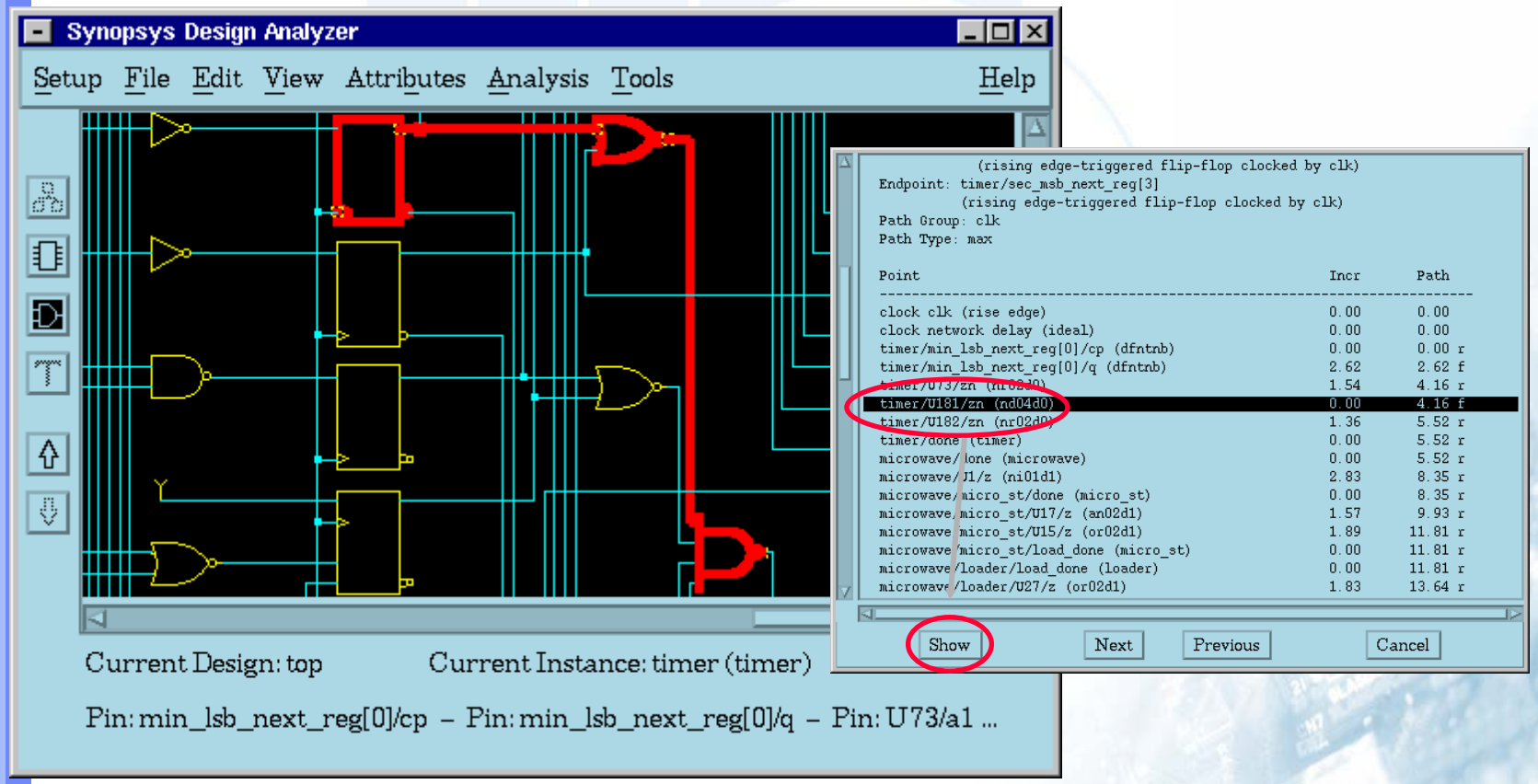
```
design_analyzer> report_timing -delay max -max_paths 1 -path short -nworst 1

****************************************
Report : timing
        -path short
        -delay max
        -max_paths 1
Design : pmult32
Version: 1999.10-5
Date   : Wed Nov 15 12:18:55 2000
****************************************

Operating Conditions: NCCOM   Library: cb35os142
Wire Load Model Mode: enclosed

  Startpoint: A2_reg[5] (rising edge-triggered flip-flop clocked by clock)
  Endpoint: out_reg[55]
            (rising edge-triggered flip-flop clocked by clock)
  Path Group: clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  pmult32            22000                 cb35os142
  pmult32_DW_bthenc_21
                     ForQA                 cb35os142
  pmult32_DW02_multp_32_10_44_0
                     4000                  cb35os142
  pmult32_DW_mtree_32_10_0
                     1000                  cb35os142
  pmult32_DW02_mult_32_10_0
                     8000                  cb35os142
  pmult32_DW01_add_41_0
                     1000                  cb35os142
  pmult32_DW01_add_64_20
                     2000                  cb35os142

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clock (rise edge)                  0.00       0.00
  clock network delay (ideal)              0.00       0.00
  A2_reg[5]/CP (decrq2)                    0.00       0.00 r
  A2_reg[5]/Q (decrq2)                     0.52       0.52 f
  ...
  out_reg[55]/D (dfnrb1)                   6.85       7.37 f
  data arrival time                                   7.37

  clock clock (rise edge)                  8.00       8.00
  clock network delay (ideal)              0.00       8.00
  out_reg[55]/CP (dfnrb1)                  0.00       8.00 r
  library setup time                      -0.15       7.85
  data required time                                  7.85
  -----------------------------------------------------------
  data required time                                  7.85
  data arrival time                                  -7.37
  -----------------------------------------------------------
  slack (MET)                                         0.47
```

Timing Report (hold time check)

```
design_analyzer> report_timing -delay min -max_paths 1 -path short -nworst 1

****************************************
Report : timing
        -path short
        -delay min
        -max_paths 1
Design : pmult32
Version: 1999.10-5
Date   : Wed Nov 15 12:28:46 2000
****************************************

Operating Conditions: NCCOM   Library: cb35os142
Wire Load Model Mode: enclosed

  Startpoint: b_reg[22] (rising edge-triggered flip-flop clocked by clock)
  Endpoint: B1_reg[11] (rising edge-triggered flip-flop clocked by clock)
  Path Group: clock
  Path Type: min

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  pmult32            22000                 cb35os142

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clock (rise edge)                  0.00       0.00
  clock network delay (ideal)              0.00       0.00
  b_reg[22]/CP (dfnrq1)                    0.00       0.00 r
  b_reg[22]/Q (dfnrq1)                     0.48       0.48 r
  B1_reg[11]/D (dfnrq1)                    0.00       0.48 r
  data arrival time                                   0.48

  clock clock (rise edge)                  0.00       0.00
  clock network delay (ideal)              0.00       0.00
  B1_reg[11]/CP (dfnrq1)                   0.00       0.00 r
  library hold time                        0.00       0.00
  data required time                                  0.00
  -----------------------------------------------------------
  data required time                                  0.00
  data arrival time                                  -0.48
  -----------------------------------------------------------
  slack (MET)                                         0.48
```

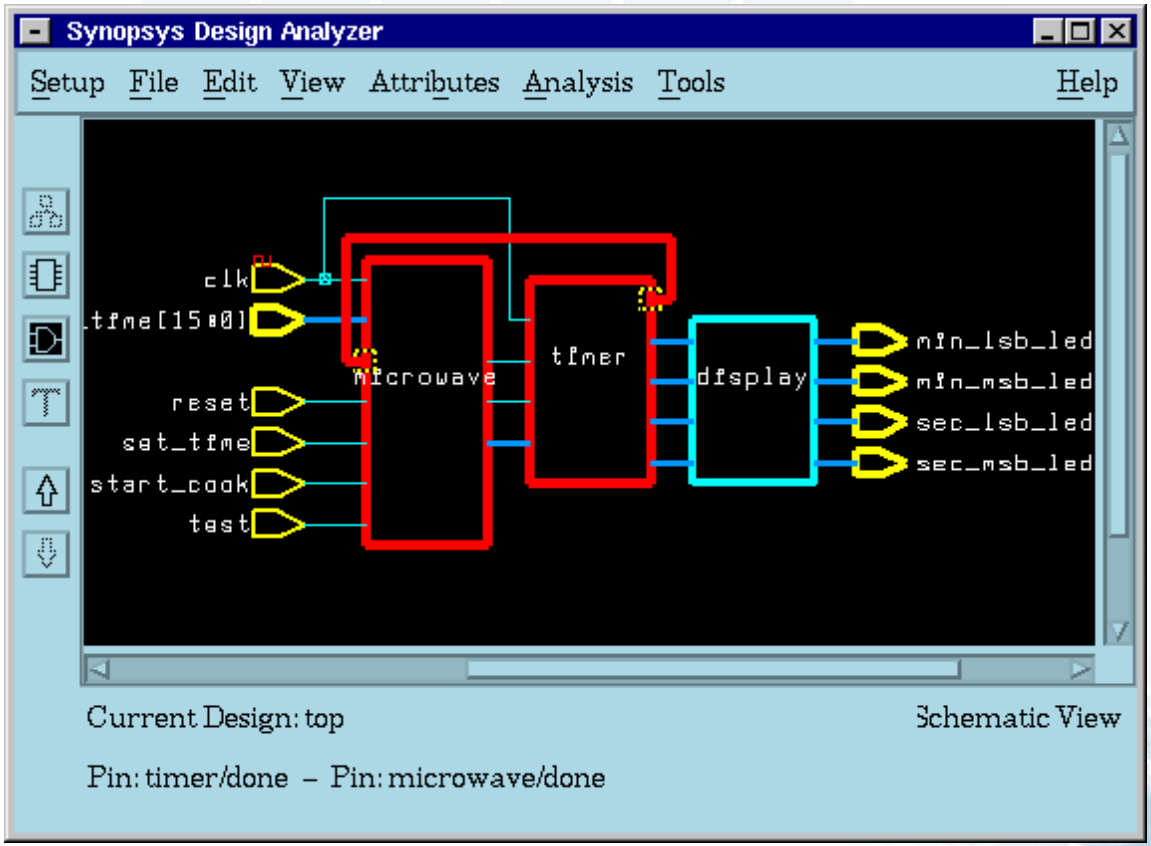
Interactive Display (1/2)

- Select a message from report, the item will highlight on the schematic (Show button)



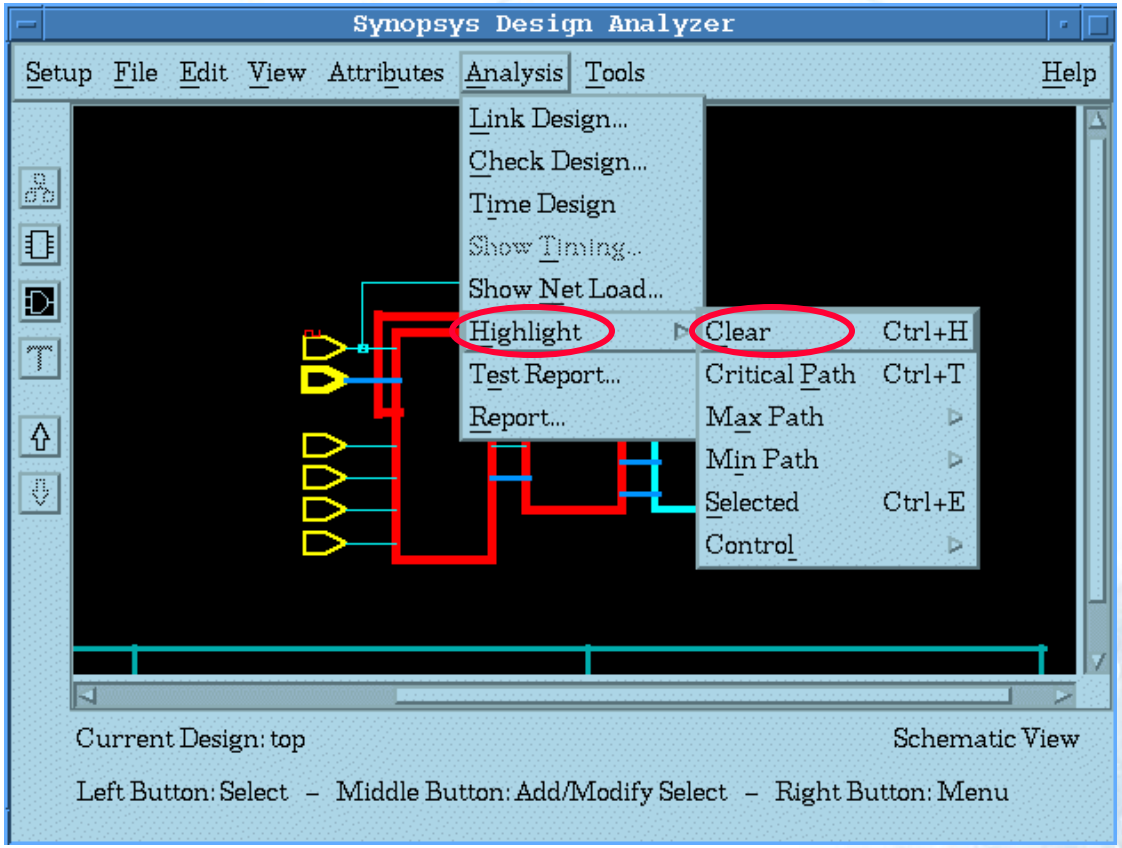
Interactive Display (2/2)

- Interactive display will across hierarchy



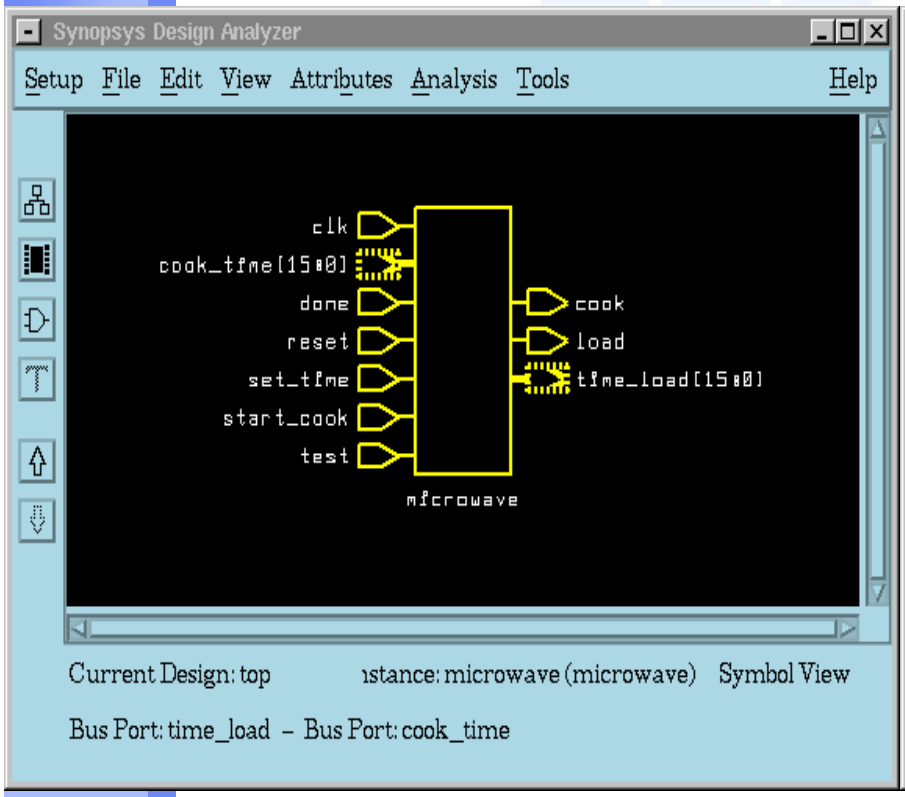
Highlight

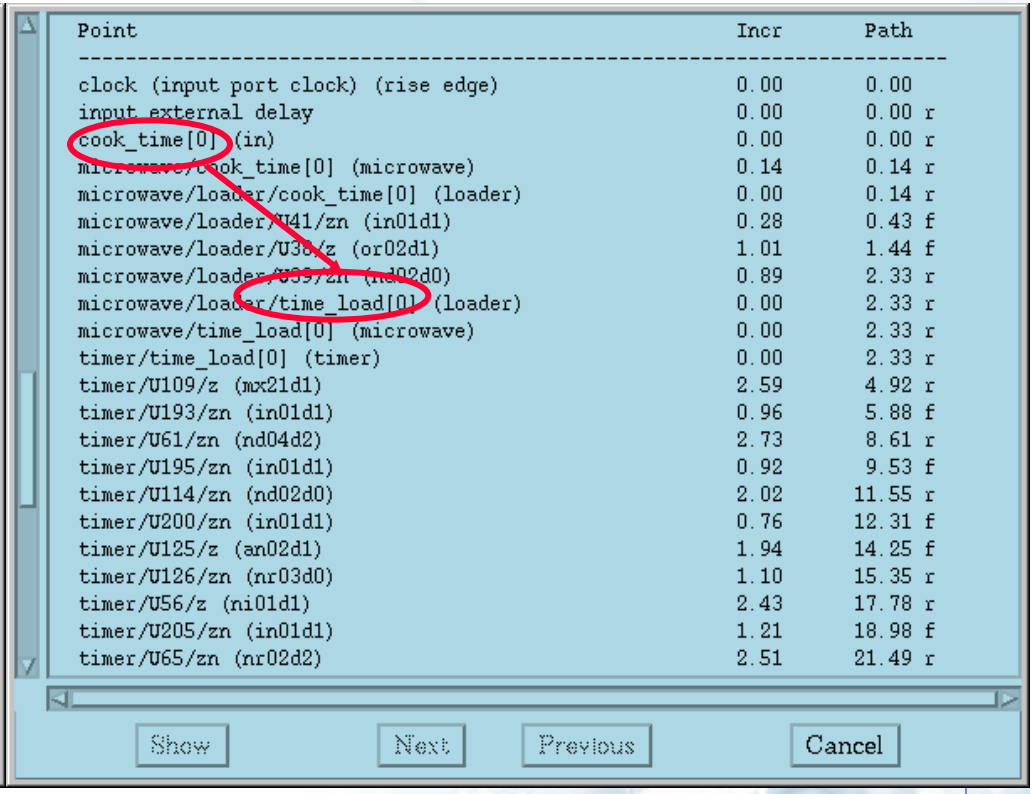
- Another way to display maximum & minimum path - Analysis/Highlight



Point Timing Report

- Point timing report shows the timing information between two selected points in the schematic
- Analysis/report → point timing





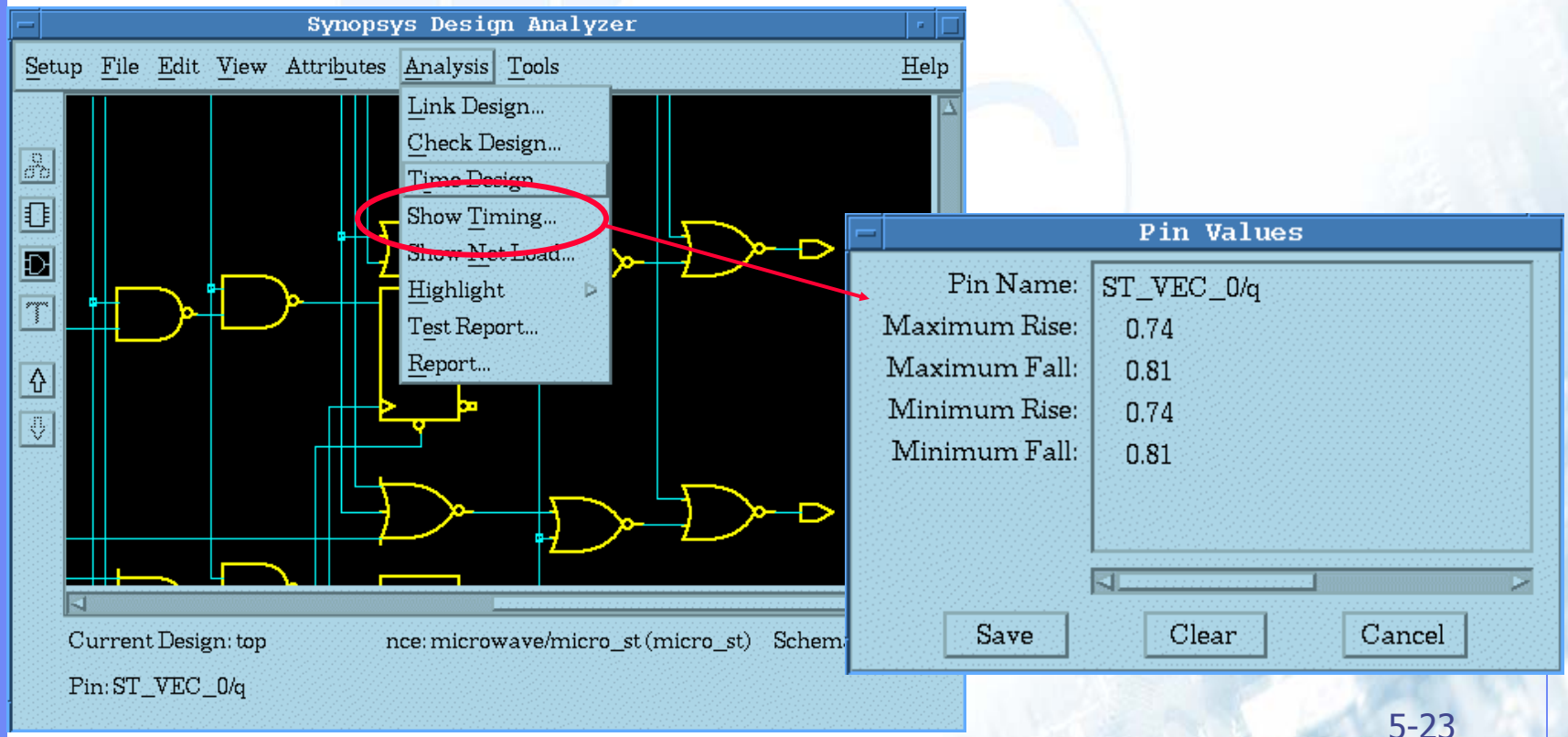
Point	Incr	Path
clock (input port clock) (rise edge)	0.00	0.00
input external delay	0.00	0.00 r
cook_time[0] (in)	0.00	0.00 r
microwave/cook_time[0] (microwave)	0.14	0.14 r
microwave/loader/cook_time[0] (loader)	0.00	0.14 r
microwave/loader/U41/zn (in01d1)	0.28	0.43 f
microwave/loader/U36/z (or02d1)	1.01	1.44 f
microwave/loader/U39/zn (nd02d0)	0.89	2.33 r
microwave/loader/time_load[0] (loader)	0.00	2.33 r
microwave/time_load[0] (microwave)	0.00	2.33 r
timer/time_load[0] (timer)	0.00	2.33 r
timer/U109/z (mx21d1)	2.59	4.92 r
timer/U193/zn (in01d1)	0.96	5.88 f
timer/U61/zn (nd04d2)	2.73	8.61 r
timer/U195/zn (in01d1)	0.92	9.53 f
timer/U114/zn (nd02d0)	2.02	11.55 r
timer/U200/zn (in01d1)	0.76	12.31 f
timer/U125/z (an02d1)	1.94	14.25 f
timer/U126/zn (nr03d0)	1.10	15.35 r
timer/U56/z (ni01d1)	2.43	17.78 r
timer/U205/zn (in01d1)	1.21	18.98 f
timer/U65/zn (nr02d2)	2.51	21.49 r

Show Next Previous Cancel

Analyze Circuit with Schematic

- To determine the time the signal arrived at pin which is selected in the schematic

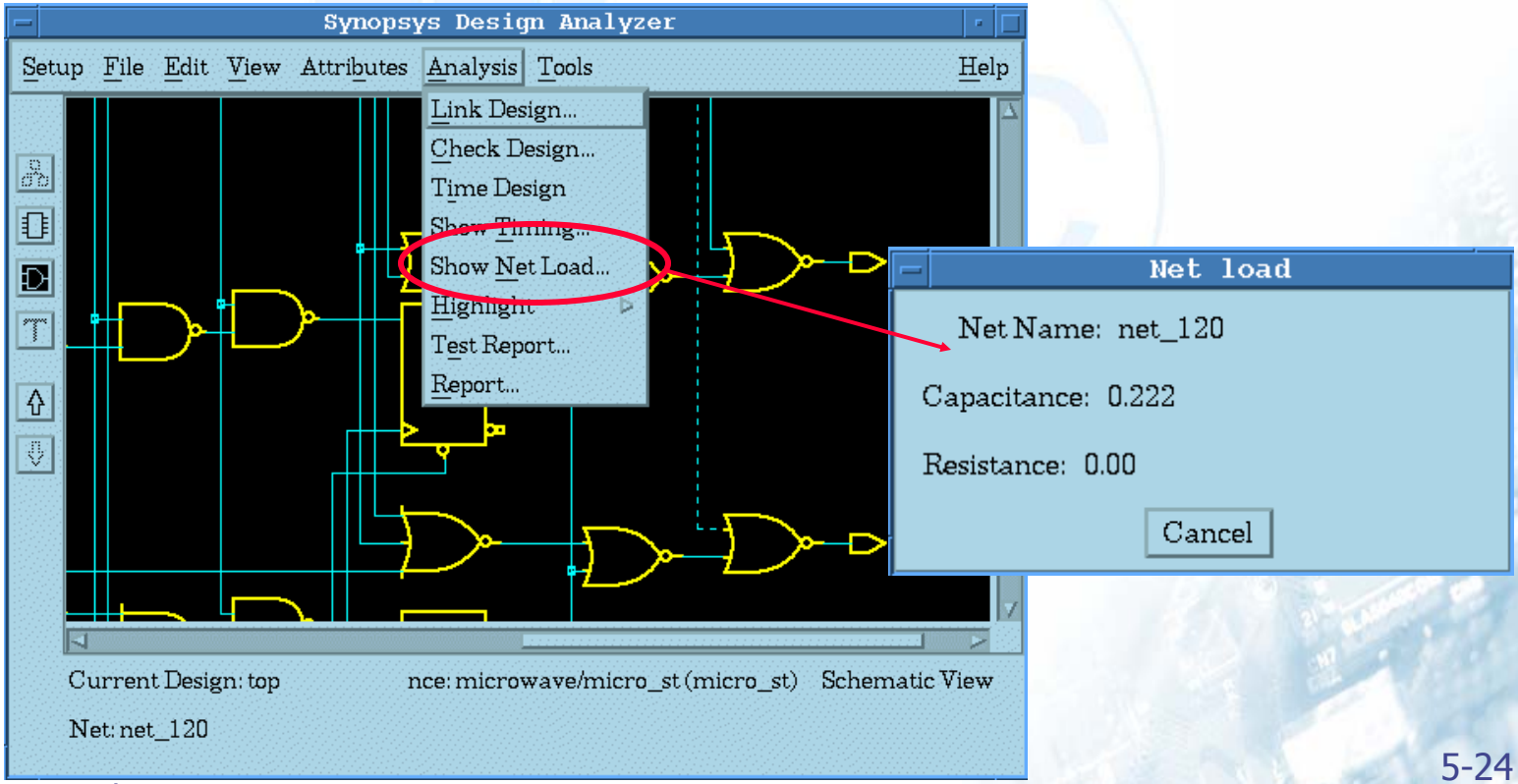
  Analysis/Show Timing



Analyze Circuit with Schematic

- To determine the net load which selected in the schematic

  Analysis/Show Net Load

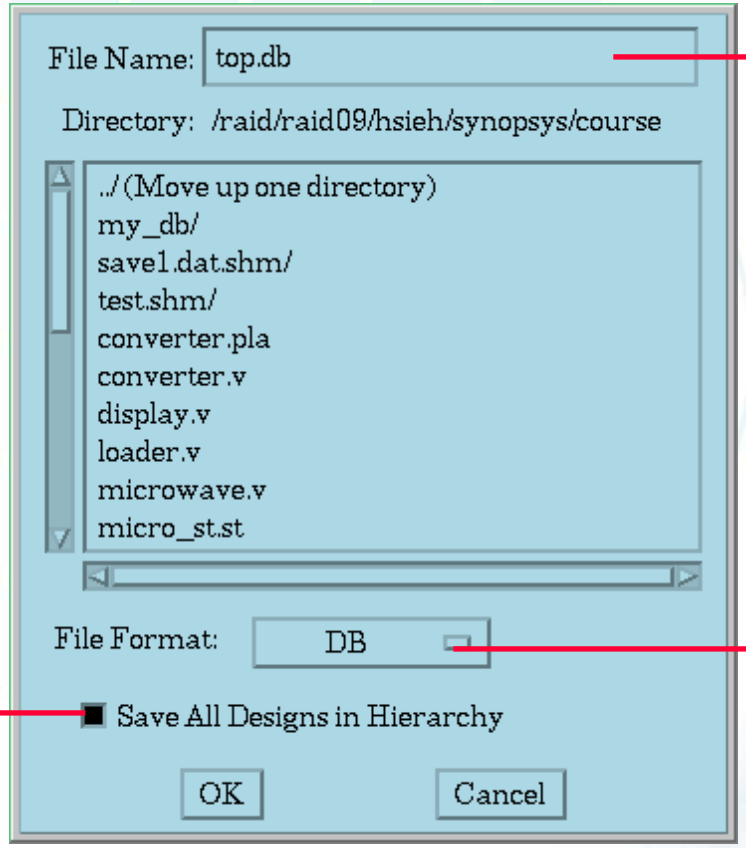


Save Design (1/3)

- Save your design to file before you quit Design Analyzer
- File/Save saves your design in the DB format
- File/Save As can save your design in other Write formats:
  - Synopsys formats
    - equation: .eq
    - state table: .st
  - Verilog: .v
  - VHDL: .vhd
  - PLA(Berkeley Espresso): .pla
  - EDIF

Save Design (2/3)

- File/Save As



File name specify

File format specify

Save design in whole hierarchy

Save Design (3/3)

- 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`

Fix Multiple Port Net

- Get rid of verilog assignment problems.
  - `set_fix_multiple_port_nets -all -buffer_constants`
  - `compile -map_effort medium`

```
assign \A[19]  = A[19];
assign \A[18]  = A[18];
assign \A[17]  = A[17];
assign \A[16]  = A[16];
assign \A[15]  = A[15];
assign ABSVAL[19] = \A[19] ;
assign ABSVAL[18] = \A[18] ;
assign ABSVAL[17] = \A[17] ;
assign ABSVAL[16] = \A[16] ;
assign ABSVAL[15] = \A[15] ;
```

```
buffda X37X ( .I(A[19]), .Z(ABSVAL[19]) );
buffd1 X38X ( .I(A[18]), .Z(ABSVAL[18]) );
buffd1 X39X ( .I(A[17]), .Z(ABSVAL[17]) );
buffd1 X40X ( .I(A[16]), .Z(ABSVAL[16]) );
buffd1 X41X ( .I(A[15]), .Z(ABSVAL[15]) );
```

Gate-Level Simulation (Verilog)

- Write out gate-level netlist (two methods)
  1. *File/Save As ➔ Verilog (for File format)*
  2. dc_shell> *write -format verilog -hierarchy –output chip.vg*
- Get SDF (two methods)
  1. *File/Save Info ➔ Design timing ➔ Select chip.sdf*
  2. dc_shell> *write_sdf -version 1.0 -context verilog chip.sdf*
- Modifiy your testfixture file

  $sdf_annotate("the_SDF_file_name",
  the_top_level_module _instance_name);

  For example: $sdf_annotate("chip.sdf", top);
- Simulation using Verilog XL

  verilog chip.vg your_testfixture.v -v your_simulation_model.v

Logic Synthesis Lab1

- Login guest account

Machine : **trainaxx**

Account : **trainaxx**

Passowrd: **traina00xx**

Lab1-1

1. Change directory ***SYNOPSYS/LAB/lab1***
   Unix%> cd ~/SYNOPSYS/LAB/lab1

   Before invoking design analyzer, please check if there is a file named"***.synopsys_dc.setup***" please take a look at this file to see what is defined in this file, then invoke the design analyzer " ***da &*** "
   Unix%> ls -al .synopsys_dc.setup
   Unix%> more .synopsys_dc.setup
   Unix%> da&

2. Open the command window by the way of design analyzer's menu bar "***setup/command window***"

3. Read the file "***lab1.v***" in
   Use the **da** menu bar "***File/Read***" to read the "***lab1.v***" in.
   Is there any problem? What is the error # ? _____

4. Invoke the Synopsys OnLine Documentation by using the command in unix shell.
   Unix%> acroread /usr/synopsys/sold/cur/top.pdf
   (Or use "***man #error_no*** or ***help #error_no"*** in command window)

5. Select the "***man Pages and Error Message***", and find the error # of step 3

6. Modify the "***lab1.v***" to fix the error

(You can use ***vi*** or other text editor)

7. Read the file "***lab1.v***" again to see if there still have any error message or warning message. If exists, what is the error or warning # ? _____

8. Use the **S**ynopsys **O**n **L**ine **D**ocumentation ***(SOLD)*** to see why this warning message occurs and modify the "***lab1.v***" again
   (Or use "***man #error_no*** or ***help #error_no"*** in command window)

9. Compile the design
   Use the ***da*** menu bar "***Tools/Design Optimization***" to synthesize your design. After synthesis, look at the ***da*** window, what happened? ➔ One more block appears. Why does this block appear?
   _____

10. Down to the "***Schematic View***" to see the result after synthesis. How many adders are used after synthesis? _____

11. Choose the other block named "***lab1_DW01_add_9_0***", down to the "***Schematic View***" to see the structure of this adder. What is this adder synthesized – ***CLA*** or ***ripple*** or other implementation form? _____

12. Back to the design view and select the "***lab1***" block. use "***analysis/report/resources***" to see what kind of adder is used by design compiler

13. Use the ***da*** menu bar "***Analysis/Report***" to see the ***timing & resources*** information of both blocks.

14. ***Set Timing Constraints*** and repeat the ***step 9-13*** again.
    In the "***Symbol View***", choose all input and output ports and in the ***da*** menu bar, choose

"Attributes/Optimization Constraints/Timing Constraint "
➔
Set the "**Maximum Delay"** to 2

15. Down to the "***Schematic View***" of adder to see the structure of this adder. What is this adder synthesized – ***CLA*** or ***ripple*** or ***clf*** or other implementation form? _____. What causes this? _____

16. How can we change this adder to CLA structure?
    Please reference the training course lecture page "***2 – 81, 82***" to reach this goal

17. After changing the implementation method, see the "***Schematic View***" & the timing report again. Is there any difference? _____

18. Follow the ***step 16-17***, change the adder implementation to ***rpl***, and see the difference on timing report

The way to use the DesignWare library above is called "***inference***", but we will find there are two floating pins in this adder– ***(Cin & Cout)***. How can we take advantage of this two pins to reduce one bit or one adder? The answer is the other way to use DesginWare library, called "***instantiation***", please reference the online documentation to do the lab below.

19. Read the file "***lab2.v***" into the design analyzer, and then compile it.

20. Down to the schematic view to see how many adders being synthesized in this example? _____

21. Take a look at the file "***lab2.v***", you will find we just have an one-bit additional input ***Cin***, and one-bit additional output Cout in this example but the synopsys uses two 8-bit adder to implement it.

This style of implementation waste lots of resources, how can we fix this mistake? (By using ***INSTANSTIATION***)

Lab1-2

Mary said "I have two kinds of designs, one contains Resource Sharing concept ***(Res_WShr.v)***, and the other has No Resource Sharing concept ***(Res_NShr.v)*** in the design. But after synthesizing these two designs, I found area in ***Res_NShr.v*** is smaller than that in ***Res_Wshr.v*** in the design."

She is very confused about this result!

1. Take a look at these two designs "***Res_Wshr.v***" and "***Res_NShr.v***". Is anything wrong in Mary's description?

   _____

2. Synthesize these two designs using ***design_compiler*** **w**ithout applying any constraints. And write down the area result using "***Analysis/Report"*** to see the area in the two designs.
   1. Res_Wshr.v_____
   2. Res_NShr.v _____

3. Use the ***da*** menu bar "***Analysis/Report***" to see the ***resources*** information of both blocks.
   What do you find?_____

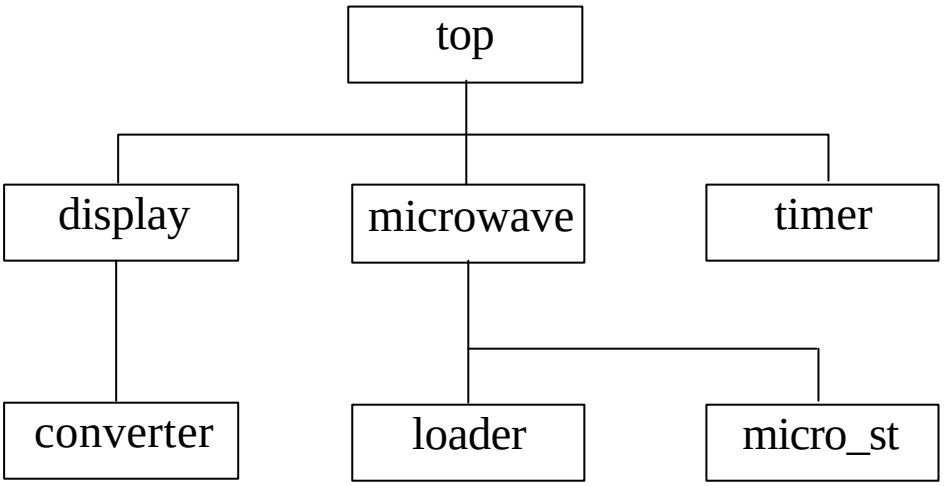
4. Modify the design ***Res_WShr.v*** by ***instantiating*** the adder ***(DW01_add)*** in designware library and re-compile this design to see the different result you get.

Logic Synthesis Lab2

Synthesis for AVANT! 0.35mm Cell Library

Introduction

The design used in this lab is a microwave timer. The design hierarchical is shown below.



The top of the hierarchical design consists of 3 blocks, ***microwave***, ***timer*** and ***display***. ***microwave*** contains a state machine, **micro_st**, which generates the control signal, and a **loader** that loads the time we want to cook, **cook_time[15:0]**, to the **timer**. **timer** will decrease the **cook_time** per second, and **display** unit uses four 7 segments LEDs to display the cook time. As the cook time reach zero, these LED will display "donE" to inform the completion of cooking.

The input and output of the top design is described below.

clk is the synchronous clock of this design.

reset is used to reset the microwave timer. If reset is zero, the design is reset to IDLE state. Until it changed to 1, then the timer start

work.

test pin is used to test if LEDs are OK. When test is 1, LED will display "8888".

cook_time is the time we want to cook.

set_time is used to set the cook time. When set_time is 1, cook_time is load into the timer.

start_cook indicates starting of cook. When start_cook is changed to 1, cooking starts, and the cook time display is decreased one per second as time passes by.

min_msb_led, min_lsb_led, sec_msb_led and **sec_lsb_led** are the outputs of BCD-to-7-segment converter, which are used to control the LEDs.

Getting Start

1. Enter the directory **SYNOPSYS/LAB/lab2**, then list the directory: **ls -al**
   ***Unix%>* cd ~/SYNOPSYS/LAB/lab2**
   ***Unix%>* ls –al**

2. Check the contents of **.synopsys_dc.setup**.
   ***Unix%>* more .synopsys_dc.setup**

3. Invoke the Design Analyzer
   ***Unix%>* design_analyzer &**

4. Open the command window
   Setup ➔ Command Window

5. Check the search path, link library, target library and symbol library. We will use **cb35os142.db** as link library and target library, **generic.sdb** as symbol library.
   Setup -> Defaults
   If it is ok, **Cancel**

Compare it with the content of .synopsys_dc.setup
more .synopsys_dc.setup

Read Design

1. Read files
File -> Read
Click on **converter.pla**
OK

Notice the icon symbol inside the box indicating PLA format. Since the files of different format can't be read at once, so we used (File➔Read). You can just read more than one file in at one time only if all the files are of the same format command.

To read the other verilog files again.

File -> Analyze
Choose "**Default**" Library
Use *left key* click on **display.v**, then use *middle key* click on **loader.v**, **microwave.v**, **micro_st.v**, **timer.v** and **top.v**
OK
File -> Elaborate
Choose "**Default**" Library
Select **top.v**
OK

Describe Design Environment

1. View the symbol view of top
Select the design **top**
Click the down arrow (on the left of ***da***), the symbol view of top will be shown.

2. Setting input driving strength

Select port **clk**

Attributes ➔ Operating Environment ➔ Drive Strength

Enter drive strength **drive_of(cb35os142 /buffd7/Z)**

Apply

Select **cook_time[15:0]**, and then use middle mouse button to select the other input port *except* **clk**

Enter drive strength **drive_of(cb35os142/buffd1/Z)**

Apply

Cancel

3. Setting output capacitance loading

Select all output ports by drag the left key

Attributes ➔ Operating Environment ➔ Load

Enter **load_of (cb35os142/buffd3/I)**

Apply

Cancel

4. Setting operating condition

Attributes ➔ Operating Environment ➔ Operating Conditions

Click on **WCCOM**

OK

Set Design Constraint

1. Specify clock

Select **clk**

Attributes ➔ Clocks ➔ Specify

set **period** as **50**

click on **Dont Touch Network**

Apply

Cancel

2. Setting input delay

Select **set_time**

Attributes ➔ Operating Environment ➔ Input Delay

select **clk**, so the **Relative To Clock** is set to **clk**, enter Max Rise and Max Fall as **10**

Apply

Cancel

3. Setting output delay

Select all the output ports

Attributes ➔ Operating Environment ➔ Output Delay

select **clk**, so the **Relative To Clock** is set to **clk**, enter Max Rise and Max Fall as **5**

Apply

Cancel

4. Setting area constraints

Attributes ➔ Optimization Constraints ➔ Design Constraints

set Max Area as **600**

Apply

Cancel

5. Check design

Analysis -> Check Design

OK

What's the warning message?_____

6. Uniquify the design converter to fix the above warning

Click on the up arrow (on the left of ***da***), the window returns to Design View

select **top**

Edit -> Uniquify -> Hierarchy

view the Design View, what's the different with before?_____

7. Generate the Port and Design reports to see whether the attributes have been properly set

Analysis -> Report

All Attributes, **Clocks**, **Design, Port**

Apply

Cancel

8. Save design and setup file

Save design

Select **top**

File -> Save As

Set File Name as **top_before_compile.db**, File Format as **DB**

Click on **Save All Designs in Hierarchy**

OK

Save setup file

File -> Save Info -> Design Setup

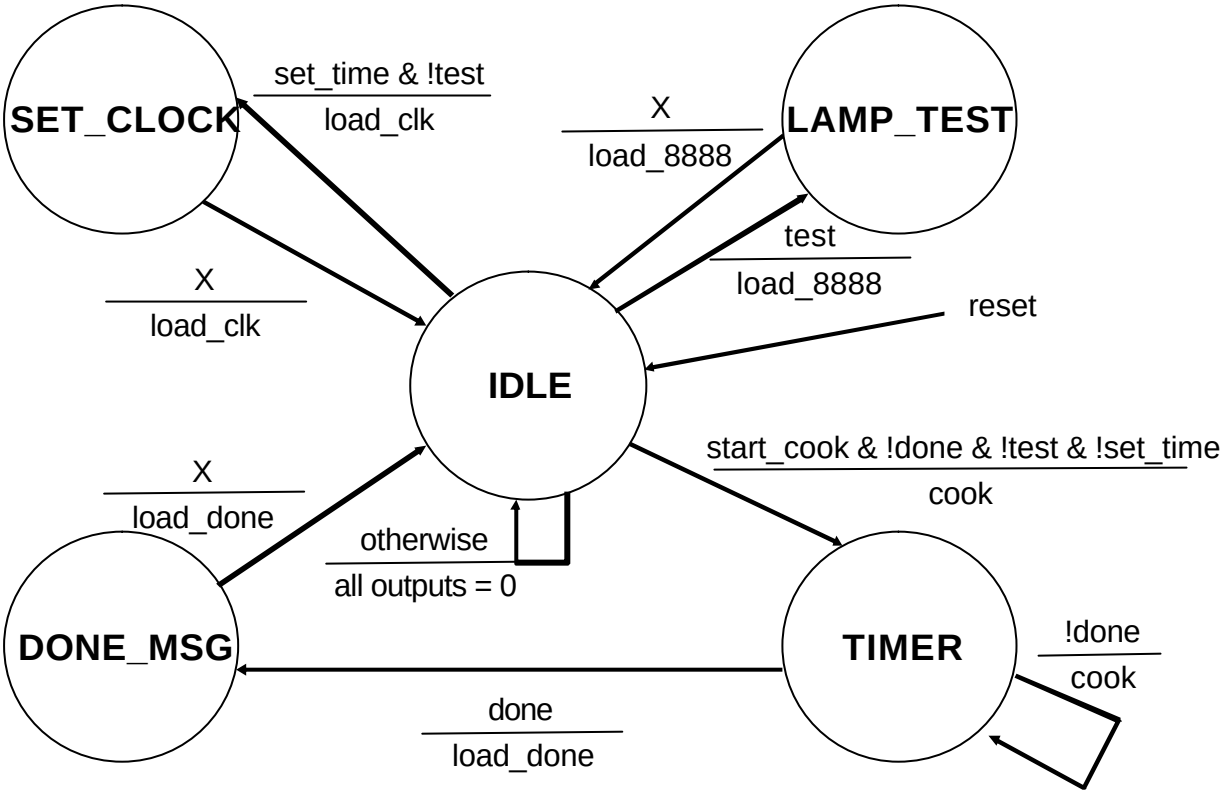
OK

Examine the top_setup.dc file in your unix shell

Unix%> **more top_setup.dc**

Finite State Machine Optimization

1. The state diagram of **micro_st.v** is shown below, take a look at it



2. Before we can extract a state table, we need to map the verilog design to gates. Therefore, compile the design **micro_st.v** first.

Select **micro_st**

Tools -> Design Optimization

OK

Examine whether the design ***micro_st*** is mapped to gate

3. Extract the finite state machine

Tools -> Finite State Machines

Extract FSM from Netlist -> Define the legal state

OK

Cancel

4. Notice the new design icon. The ***micro_st*** design is now in state table format

Save it in the state table format as **micro_st.st** and examine the file

File -> Save As

Set File Name as **micro_st.st**, File Format as **State Table**

OK

Examine the micro_st.st in your Unix shell

Unix%> **more micro_st.st**

5. Encode the state machine with the encodings that were specified in the **micro_st.v**

Tools -> Finite State Machines

State Encodings

OK

Compile

OK

Generate the report for area and critical path

Analysis -> Report

Area, Timing

Apply

Area?_____, Critical Path?_____ (See **Data Arrival Time**)

Cancel

6. To try another encoding style, we have to return to state table format. so read in the micro_st.st file, then try the one-hot encoding

File -> Read

Click on **micro_st.st**

OK

Select **micro_st** in the Design View
Tools -> Finite State Machines
State Encodings
One Hot
OK
Compile
OK

Generate the report for area and critical path
Analysis -> Report
Area, Timing
Apply
Area?_____, Critical Path?_____
Cancel

7. Now try the automatic state optimization, follow above procedures and read the micro_st.st again
Tools -> Finite State Machines
State Encodings
Clear All
OK
Cancel

We don't optimize the design micro_st now, we'll optimize it with the other part of the design later.

Compile Design

1. Suppose we want to disable Design Compiler from using register sdnrb1 and sdprb1
In the command window, enter
design_analyzer> **dont_use {cb35os142/sdnrb1, cb35os142/sdprb1}**

2. Compile design

Select **top**

Tools -> Design Optimization

set Map Effort as **Medium**

OK

Examine which designs were optimized? this is what known as "Hierarchical Compile"

3. Explore the schematic

Double click on the design until you have down to the Schematic View of the bottom cell

4. Work with some view commands

View -> Recreate

View -> Zoom In

View -> Zoom Out

View -> Full View

5. Notice that the components drawn do not show instance names, to display instance name

View -> Style

Click **cell_name_layer**

Visible : **on**

Apply

Cancel

Zoom in to see if you can see the instance name

6. Return to the Design View

Click the up arrow (on the left of ***da***), until reach the Design View

Analysis

1. Check the constraints, area, timing

    Select **top**

    Analysis ➔ Report

    Area, Constraints, Timing

        Set Options : Constraint Options : All Violations

            OK

        Apply

        Are you meeting your constraints?_____

        What's the area of this design?_____

        Which is the critical path?_____

        Cancel

2. Examine the critical path

    Click the schematic button (on the left), will show the schematic view of top

    Analysis ➔ Highlight ➔ Critical Path

    Remove highlight

    Analysis ➔ Highlight ➔ Clear

3. Generate another version of a timing report

    Analysis -> Report

        Timing

        Set Options

            Report Points : **Only End Points**

            Max Paths to Show : **7**

            OK

        Apply

        Examine if there are just one critical path

        Cancel

4. Go back Design View, generate report of **Reference**, **Hierarchy** for top

5. Generate report of **FSM** for micro_st

Save File

1. Save compiled design as db file and verilog netlist

File -> Save As

Set File Name as **top_compile.db**, File Format as **DB**

Click on **Save All Designs in Hierarchy**

OK

File -> Save As

Set File Name as **top_compile.v**, File Format as **Verilog**

Click on **Save All Designs in Hierarchy**

OK

Check if there is any warning message in the command window

2. Examine the verilog file you just created

In your unix window, type **more top_compile.v**

Unix%> **more top_compile.v**

Quit Design Analyzer

. Quit the design analyzer

File -> Quit

Do you really want to quit the Design Analyzer?

OK