Digital Circuits and Systems Lecture 10 Synthesis

Tian Sheuan Chang

SYNTHESIS OVERVIEW

Slides from Prof. Hung Ming Chen

HDL-Based Design Flow

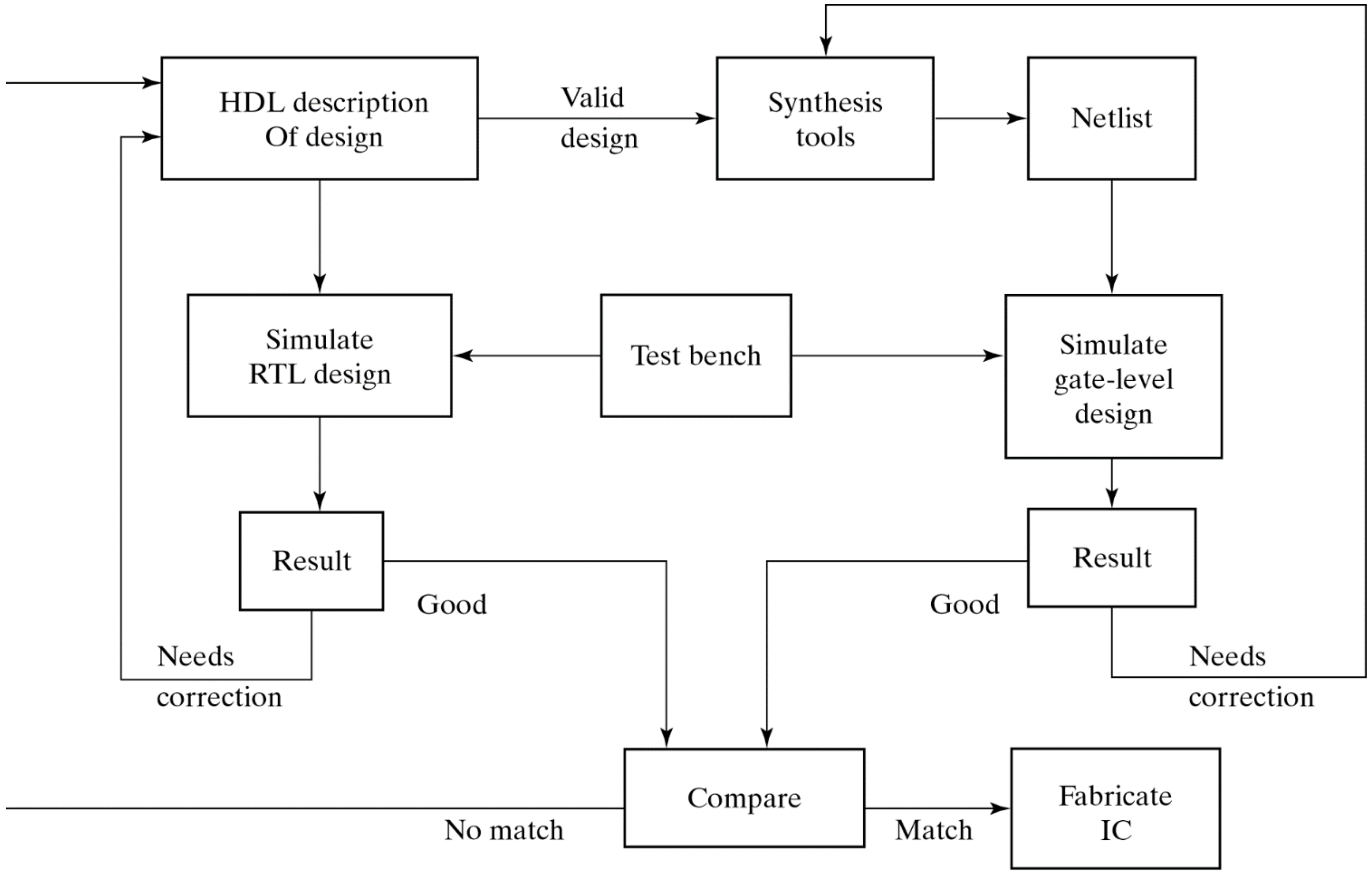
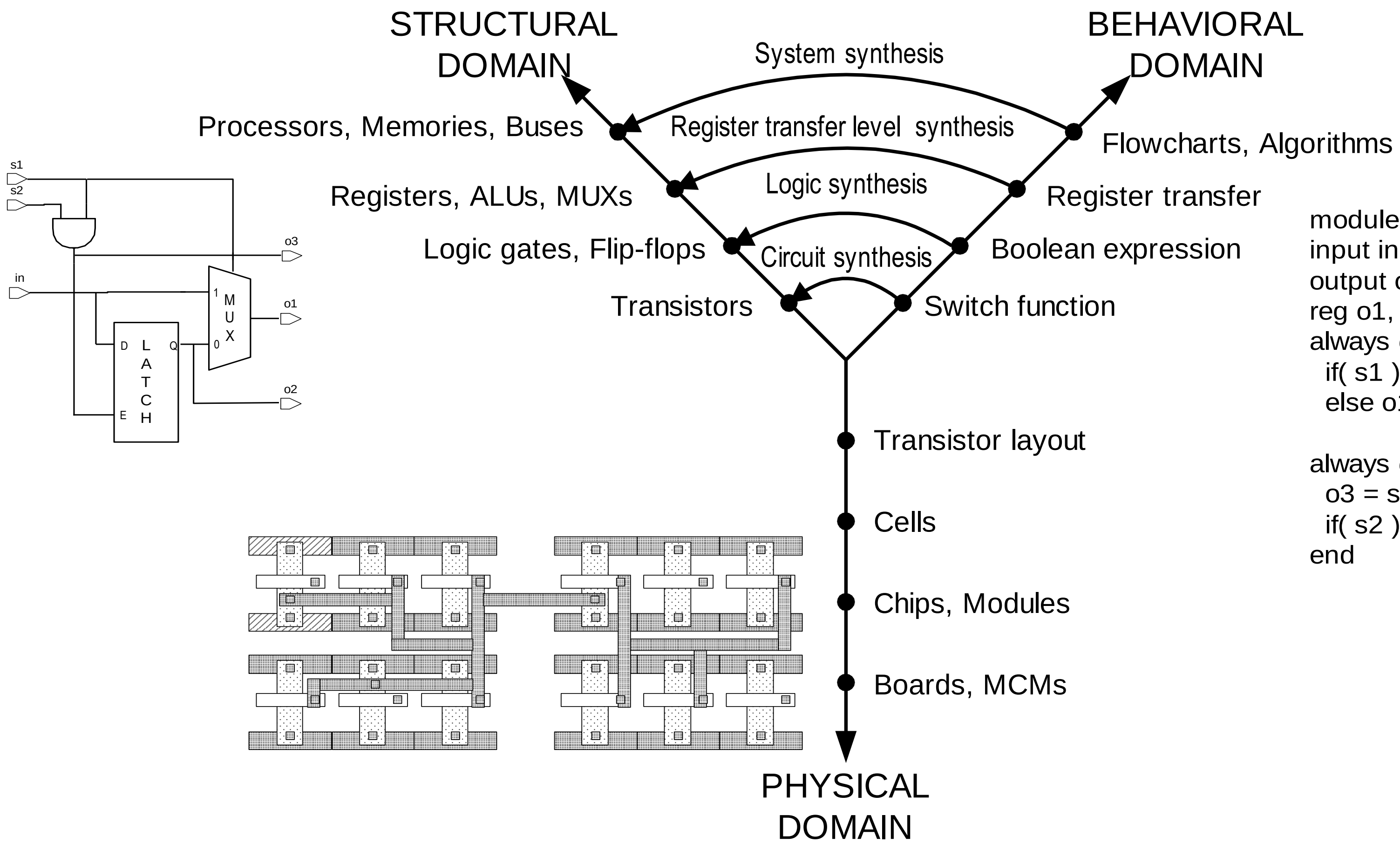


Fig. 8-1 Process of HDL Simulation and Synthesis

Levels of Design

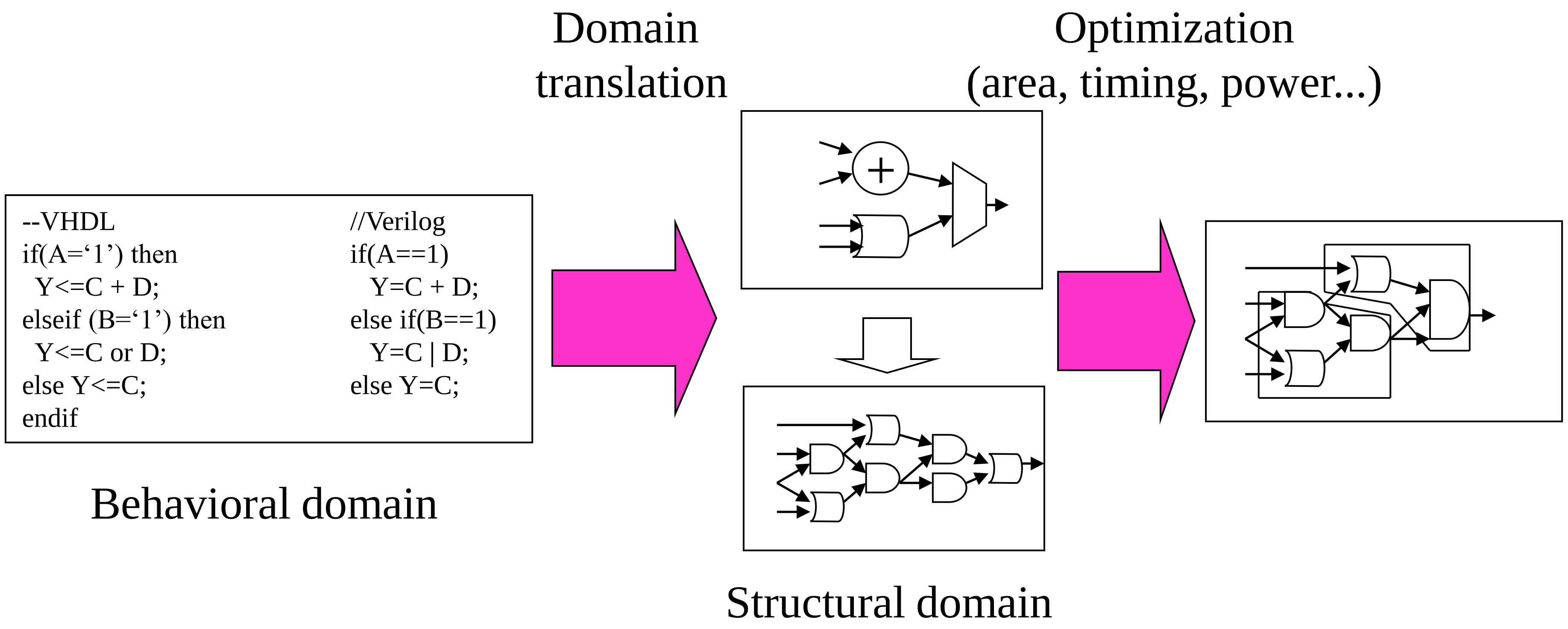


```
module EXP(in, s1, s2, o1, o2, o3);
input in, s1, s2;
output o1, o2, o3;
reg o1, o2, o3;
always @ (in or s1 or s2)
  if( s1 ) o1 = in;
  else o1 = o2;

always @ ( s1 or s2 or o1) begin
  o3 = s1 & s2;
  if( s2 ) o2 = o1;
end
```

Register Transfer Level Synthesis

- Synthesis = Domain Translation + Optimization



```
--VHDL
if(A='1') then
  Y<=C + D;
elseif (B='1') then
  Y<=C or D;
else Y<=C;
endif
```

```
//Verilog
if(A==1)
   Y=C + D;
else if(B==1)
   Y=C | D;
else Y=C;
```

Two-Level Logic Optimization

- Two-level logic representations
  - Sum-of-product form
  - Product-of-sum form
- Two-level logic optimization
  - Key technique in logic optimization
  - Many efficient algorithms to find a near minimal representation in a practical amount of time
  - In commercial use for many years
  - Minimization criteria: number of product terms
- Example: $F = XYZ + X\bar{Y}\bar{Z} + X\bar{Y}Z + \bar{X}YZ + XY\bar{Y}Z$

$$\Downarrow$$

$$F = X\bar{Y} + YZ$$

Multi-Level Logic Optimization

- Transforms a combinational circuit to meet performance or area constraints
  - Two-level minimization
  - Common factors or kernel extraction
  - Common expression re-subsitution
- In commercial use for many years
- Example:

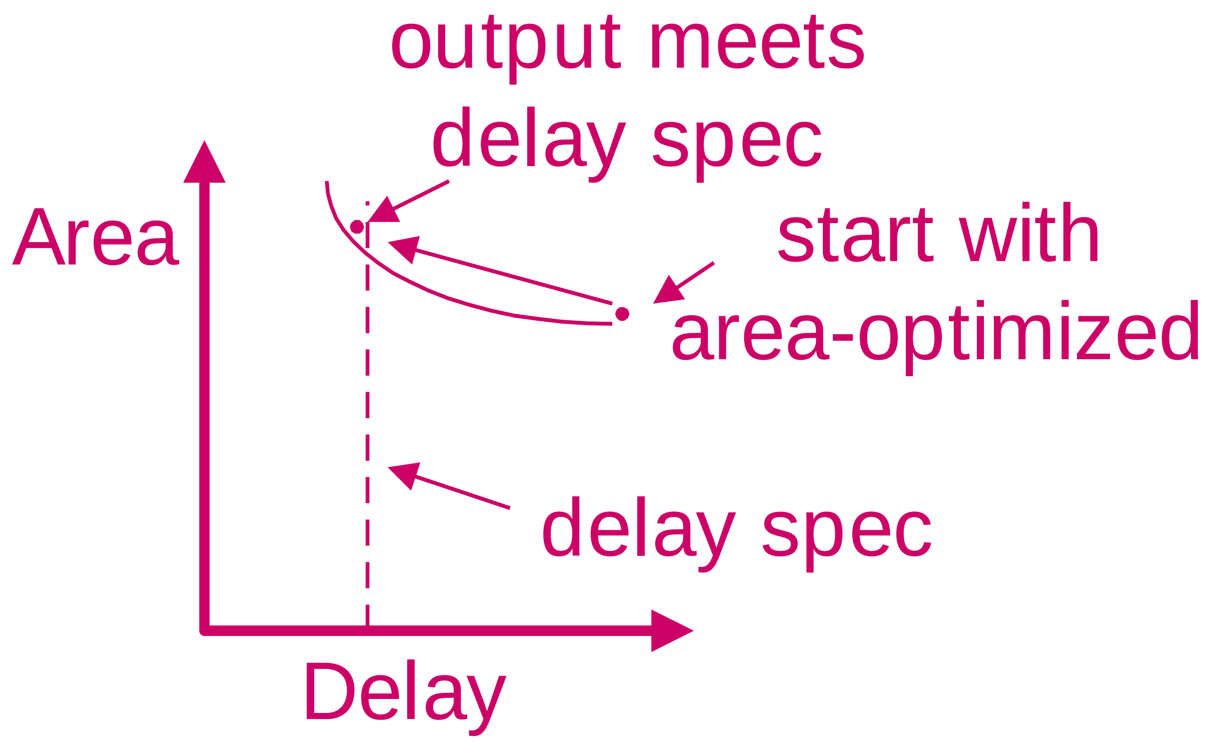
$$\begin{aligned} f1 &= abcd + abce + a\bar{b}c\bar{d} + a\bar{b}\bar{c}\bar{d} + \\ &\quad \bar{a}c + cdf + ab\bar{c}\bar{d}\bar{e} + a\bar{b}\bar{c}d\bar{f} \\ f2 &= bdg + \bar{b}dfg + \bar{b}\bar{d}g + b\bar{d}eg \end{aligned} \quad \Rightarrow \quad \begin{aligned} f1 &= c(\bar{a} + x) + a\bar{c}\bar{x} \\ f2 &= gx \\ x &= d(b + f) + \bar{d}(\bar{b} + e) \end{aligned}$$

Technology Mapping

- Goal: translation of a technology independent representation (e.g. Boolean networks) of a circuit into a circuit in a given technology (e.g. standard cells) with optimal cost
- Optimization criteria:
  - Minimum area
  - Minimum delay
  - Meeting specified timing constraints
  - Meeting specified timing constraints with minimum area
- Usage:
  - Technology mapping after technology independent logic optimization
  - Technology translation

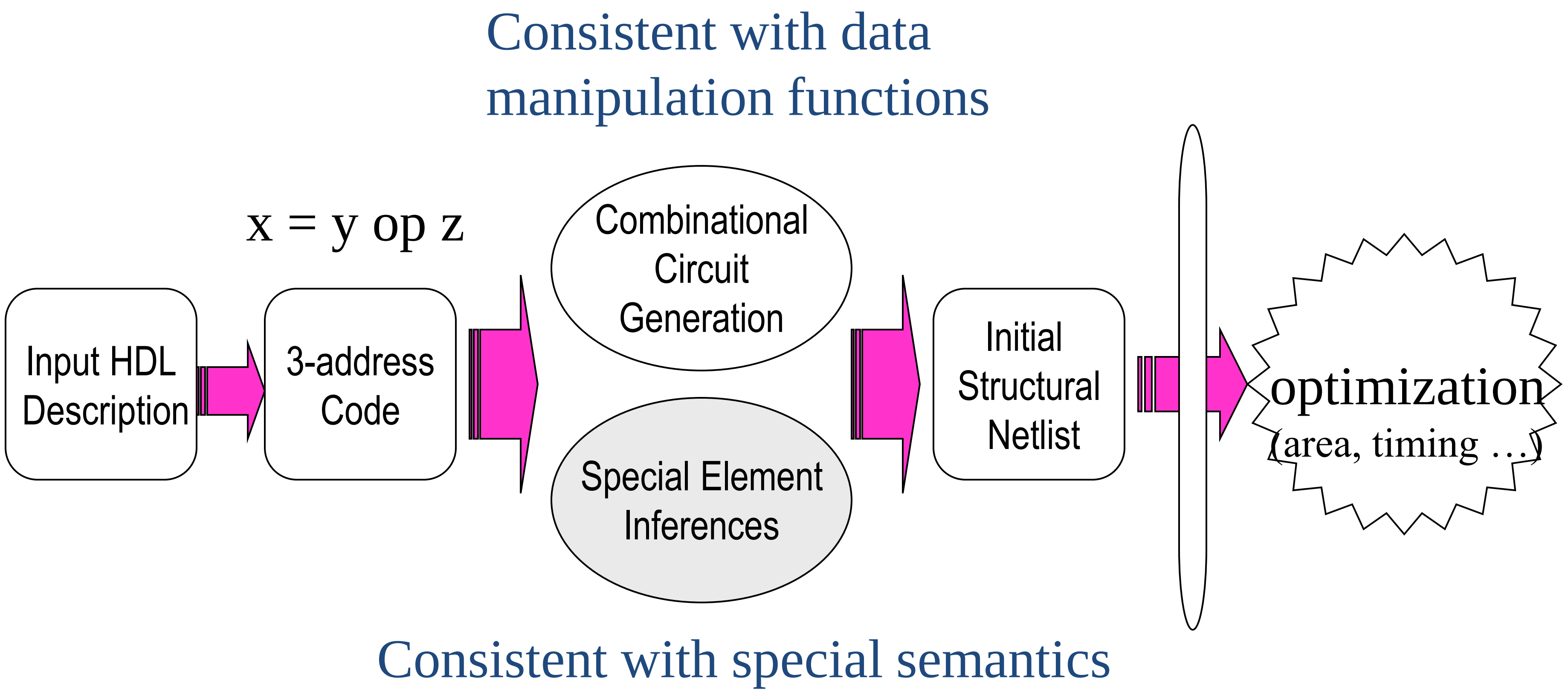
Timing Optimization

- There is always a trade-off between area and delay
- Optimize timing to meet delay spec. with minimum area



RTL SYNTHESIS

Domain Translation


Consistent with data manipulation functions
x = y op z
Input HDL Description
3-address Code
Combinational Circuit Generation
Special Element Inferences
Initial Structural Netlist
optimization
(area, timing …)
Consistent with special semantics

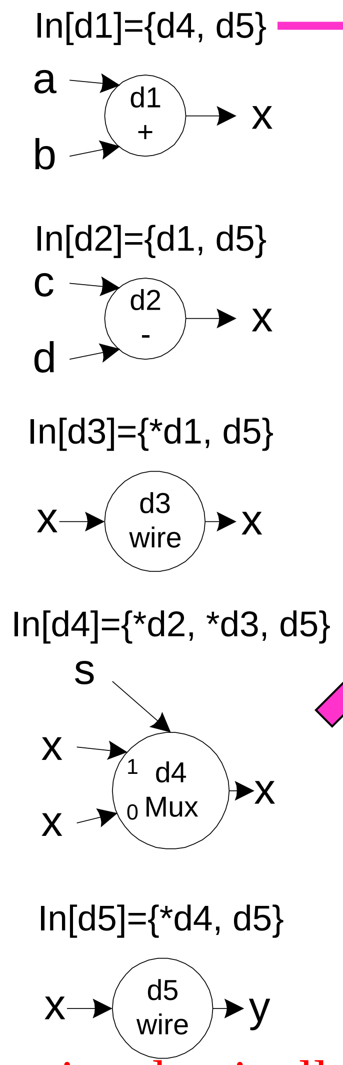
Combinational Circuit Generation

```
always @ (x or a or b or c or d or s)
begin
/*d1*/    x = a + b;
/*d2*/    if( s ) x = c - d;
/*d3*/    y = x;
end
```

Input HDL

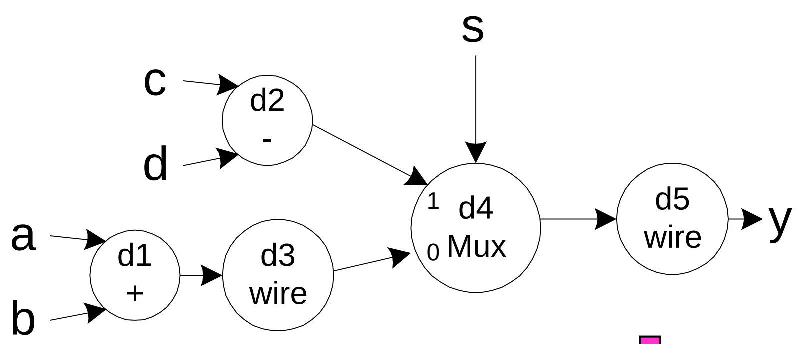
```
always @ (x or a or b or c or d or s)
begin
/*d1*/    x = a + b;
/*d2*/    if( s ) x = c - d;
/*d3*/    else x = x;
/*d4*/    x = s mux x;
/*d5*/    y = x;
end
```

Modified
3-address code

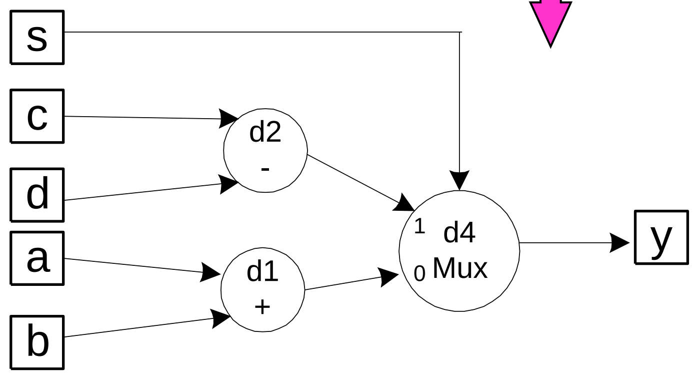


computed by control/ data flow analysis

Functional unit allocation



Interconnection binding



Final result

Special Element Inferences

- Given an HDL at RTL, three special elements need to be inferred to keep the special semantics in the HDL.
  - Latch (D-type) inference
  - Flip-Flop (D-type) inference
  - Tri-state buffer inference

```
reg Q;
always@(D or en)
   if(en) Q = D;
```

Latch is needed!!

```
reg Q;
always@(posedge clk)
    Q <= D;
```

Flip-flop is needed!!

```
reg Q;
always@(D or en)
   if(en) Q = D;
   else    Q = 1'bz;
```

Tri-state buffer is needed!!

Typical Latch Inference

- Conditional assignments are not completely specified
  - Check if the else-clause exists
- Outputs conditionally assigned in an if-statement are not assigned before entering or after leaving the if-statement

```
always@(D or S)
if(S) Q = D;
```

$\Rightarrow$ Infer latch for Q

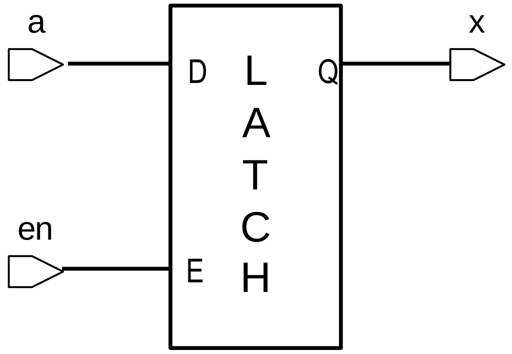
```
always@(S or A or B)
begin
Q = A;
if(S) Q = B;
end
```

$\Rightarrow$ Do not infer latch for Q

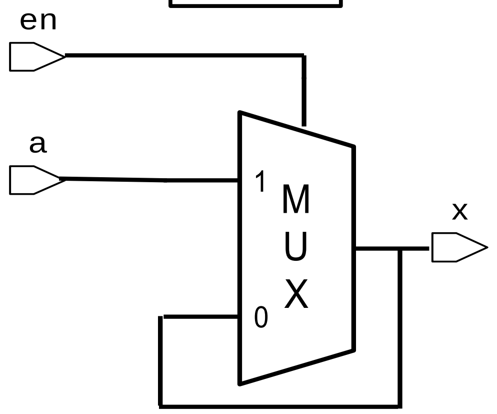
Typical Latch Inference

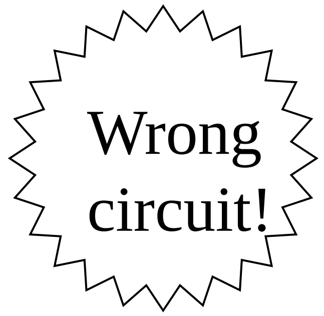
- Unreasonable limitations on coding style

```
always@(a or en)
  if(en) x=a;
```



```
always@(a or en)
  if(en) x=a;
  else x=x;
```





Latch description

Terminology

- Clocked statement: edge-triggered always statement
  - Simple clocked statement
  - e.g., **always @ (posedge clock)**
  - Complex clocked statement
  - e.g., **always @ (posedge clock or posedge reset)**
- **Flip-flop inference** must be conducted when synthesizing the clocked statements

Typical Flip-flop Inference on Complex Clocked Statements

- Require the following syntactic template
  - An if-statement immediately follows the always statement
  - Each variable in the event list except the clock signal must be a selective signal of the if-statement
  - Assignments in the blocks B1 and B2 must be constant assignments (e.g., x=1, etc.)

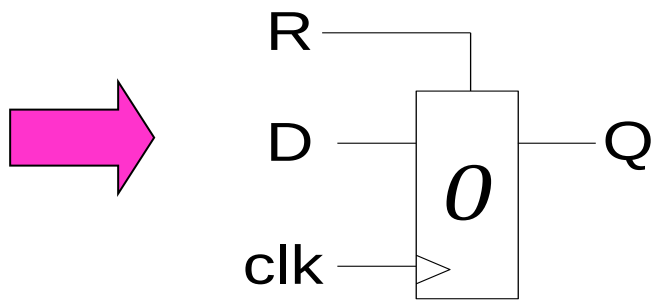
```
always @ (posedge clock or posedge reset or negedge set)

                if(reset) begin B1 end
                else if ( !set) begin B2 end
                else begin B3 end
```

Typical Flip-flop Inference on Complex Clocked Statements

- An example

```
always @ (posedge clk or posedge R)
if(R) Q = 0;
else Q = D;
```



- Disadvantages
  - Syntactic template results in unreasonable limitations in coding style

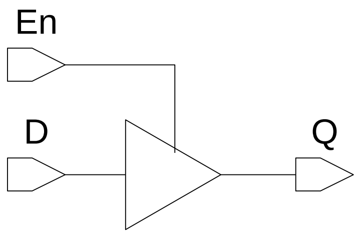
```
always @ (posedge clk or posedge R)
begin
Q = D;
if(R) Q = 0;
end
```

Non-synthesizable

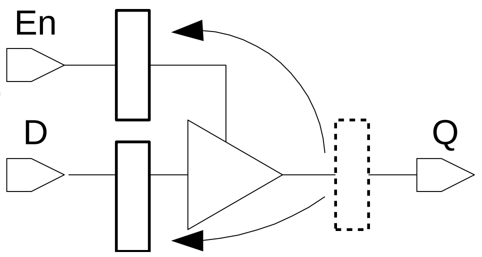
Typical Tri-state Buffer Inference

- If a data object Q is assigned a high impedance value ‘Z’ in a multi-way branch statement (if, case, ?:)
  - Associated Q with a tri-state buffer
  - If Q associated with a tri-state buffer has also a memory attribute (latch, flip-flop)
    - Real hardware cannot propagate Hi-Z (Hi-Z propagation problem)
    - Place two memory elements to the control and the data inputs of tri-state buffer

```
reg Q;
always @ (En or D)
if(En) Q = D;
else Q = 1'bz;
```



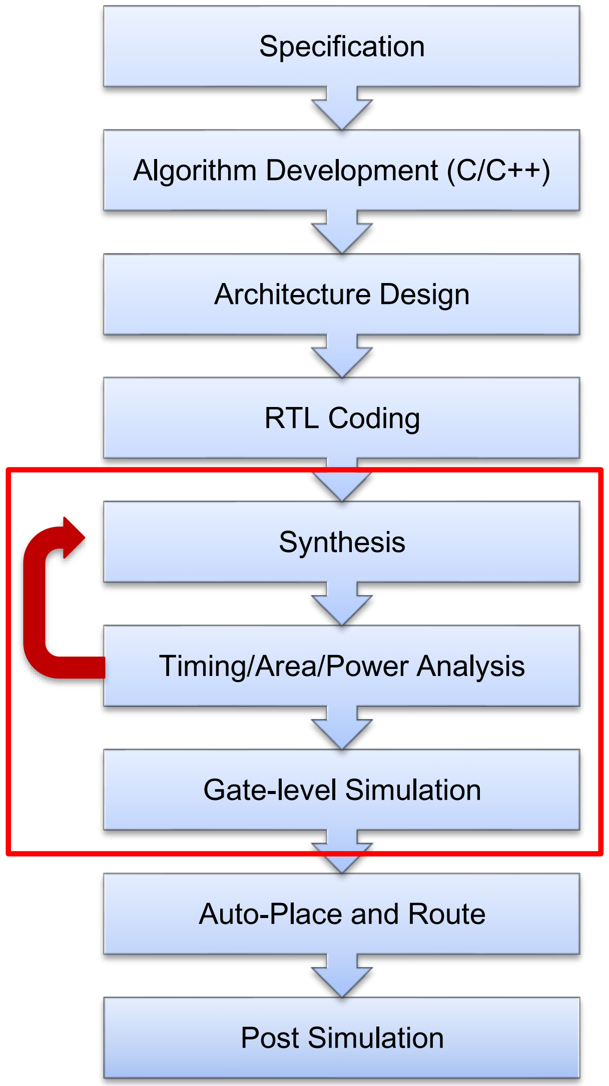
```
reg Q;
always @ (posedge clk)
if(En) Q = D;
else Q = 1'bz;
```



SYNTHESIS SCRIPTS

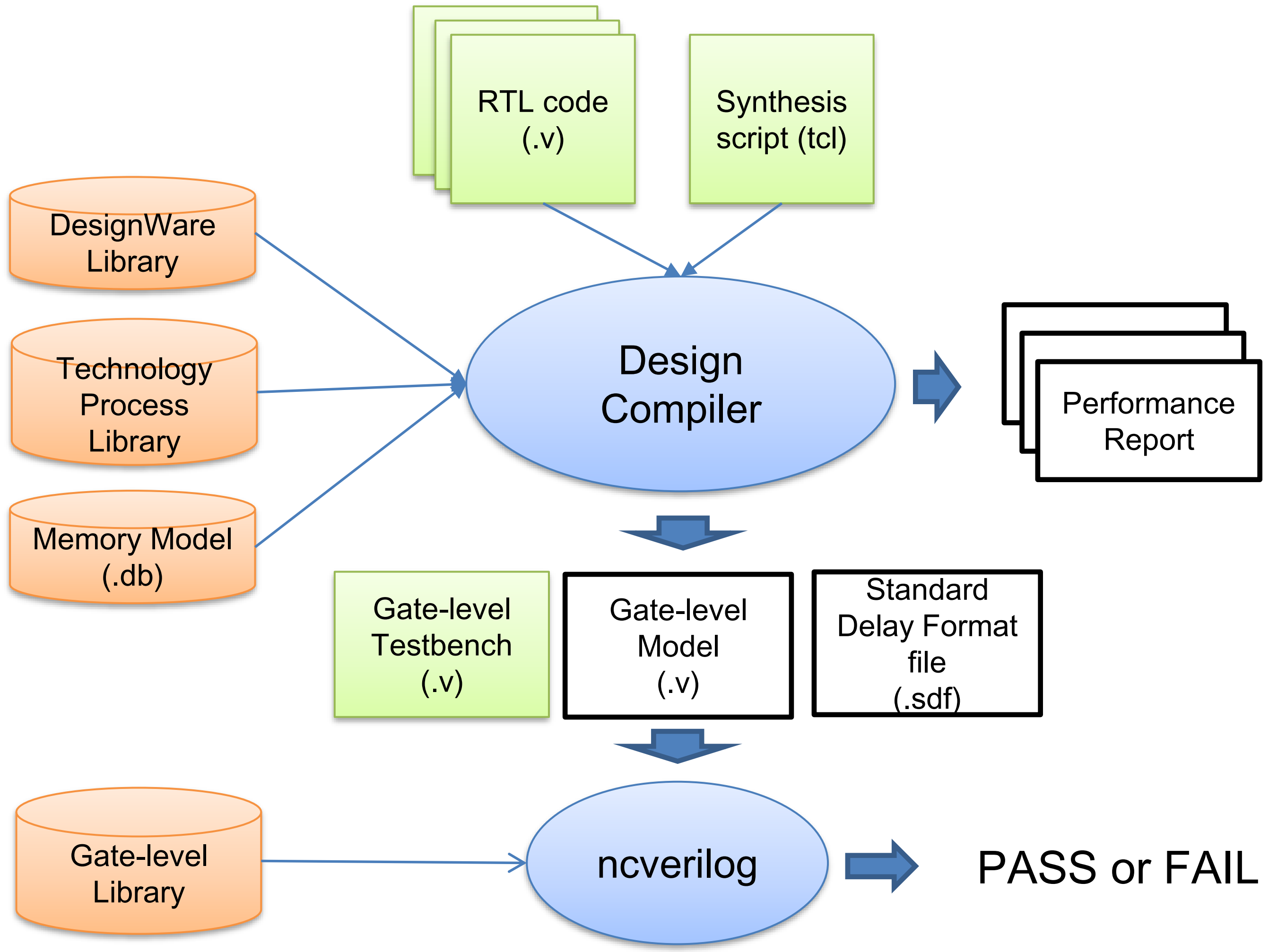
Synthesis in design flow

- Synthesis
  - Tool: Design compiler (synopsys)
  - Goal
    - Model translation from RTL to gate-level
    - Technology mapping
    - Constrained optimization
- Timing/area/power analysis
  - Tool: Design Compiler, PrimeTime, PrimePower (synopsys)
  - Goal
    - Hardware performance verification
- Gate-level simulation
  - Tool: ncverilog (cadence)
  - Goal
    - Functionality verification



Data flow in Design Compiler

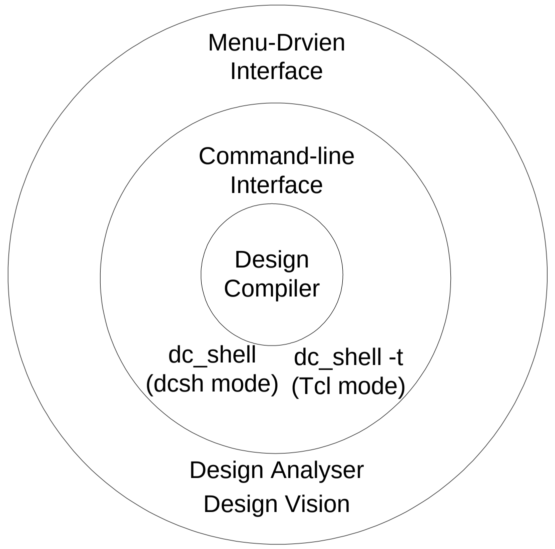
- What you need to prepare:
  - RTL codes
  - Synthesis script
  - Gate-level testbench



Introduction to Design Compiler

- Design Compiler (dc)
  - Kernel of synthesis tool
- Command-line interface
  - Easy to iterative compiling
  - dc_shell: original command of DC
  - dc_shell-t: dc_shell + tcl (tool command language)
- Menu-driven interface
  - Friendly GUI interface
  - Design Vision (dv) supports dc_shell and dc_shell-t
  - Design Analyser (da) only supports dc_shell
  - Open tools by command
    - dv &
    - da &

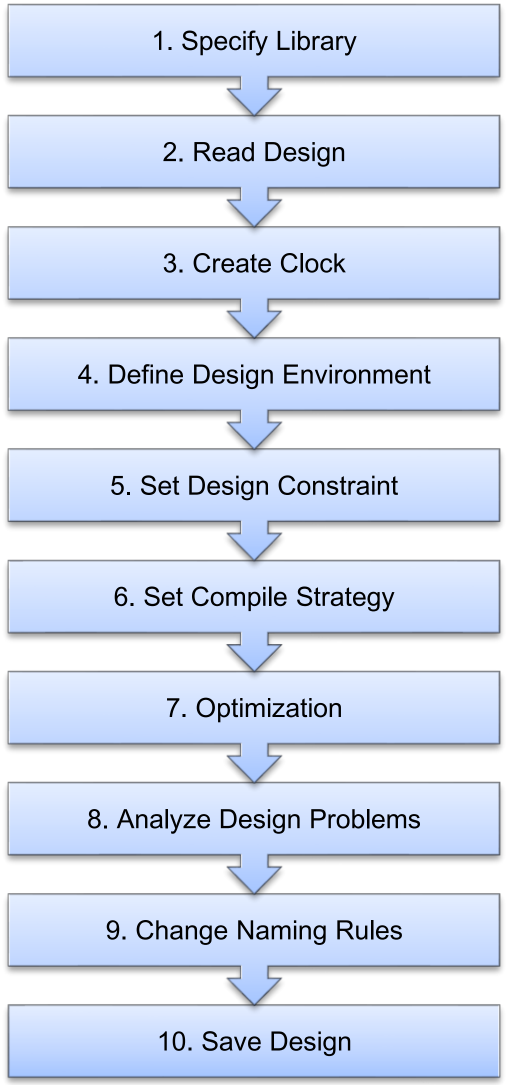
Before learning DC, you should learn the tcl first.



Synthesis Script in DC

Flow

- First, select a semiconductor vendor
  - TSMC
  - UMC
  - others
- Default setting: .synopsys_dc.setup
  - Search directory
    - Synopsys root directory
    - Home directory
    - Working directory (highest priority)
  - 3 files are executed automatically.
  - Viewed by command ls –la



1. Specify library

- Search path
  - lists all the related directories of design and libraries.
  - Syntax: set search_path {"…"}
- Link library
  - lists all technology process libraries with various timing condition.
  - Syntax: set link_library {"wc.db" "bc.db"}
- Target library
  - selects libraries from link_library.
  - Syntax: set target_library {"wc.db" "memory.db"}
- Symbol library
  - defines symbols of schematic view for DV or DA.
  - Syntax: set symbol_library file.sdb
- DesignWare library
  - defines built-in operators in Synopsys
  - Syntax: set synthetic_library {"dw_foundation.sldb"}

2. Read design

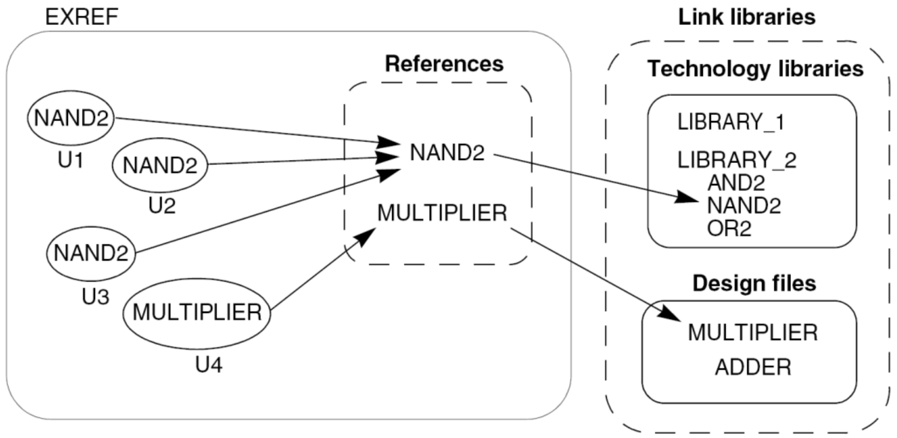
- Supported format
  - Verilog, VHDL, SystemVerilog
  - Synopsys internal database format (.db, .ddc)
  - Berkeley Escpresso format (.pla)
- Method1:
  - analyze (only HDL)
    - Read HDL source files, check errors.
    - Create and store HDL-independent intermediate objects.
  - elaborate (only HDL)
    - Translate intermediate objects into technology-independent design (GTECH).
    - Load parameters in HDL source files.
    - Replace DesignWare components.
    - Execute link command to resolve design references.
- Method2:
  - read_file
    - Read files with selected format.
    - Execute the same steps in analyze and elaborate (without link and loading param.).
    - Do not create intermediate objects (default).
    - Ex1: read_file –rtl -format verilog $RTL
    - Ex2: read_file –netlist -format verilog $GATE_LEVEL

Set current design

- Only one design can be set as current design for optimization.
- Setting current design ways:
  - Automatic setting after command
    - read_file
    - elaborate
  - Direct command:
    - current_design design_name

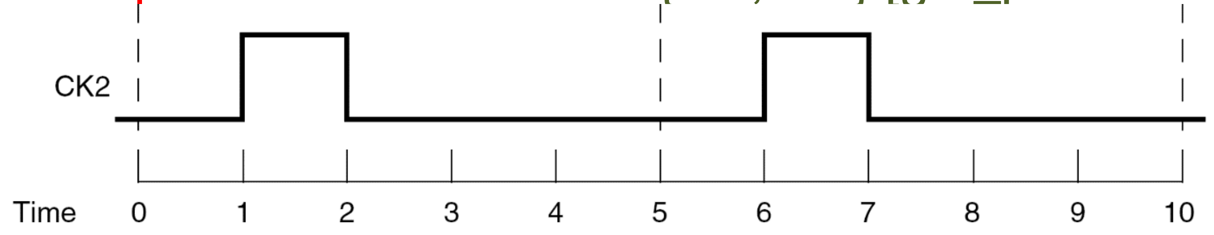
Link design

- Liking design (resolving references) is to connect all library components and design its references.
- Command:
  - link
- Performed steps:
  - libraries to reference
    - local_link_library (highest priority)
    - link_library
    - search_path (lowest priority)
  - reference to design

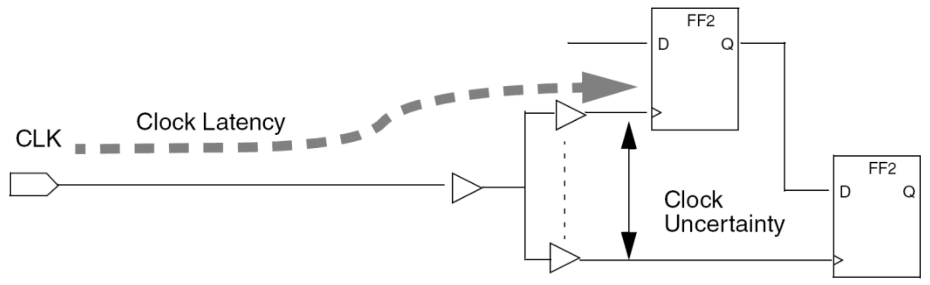


3. Create clock

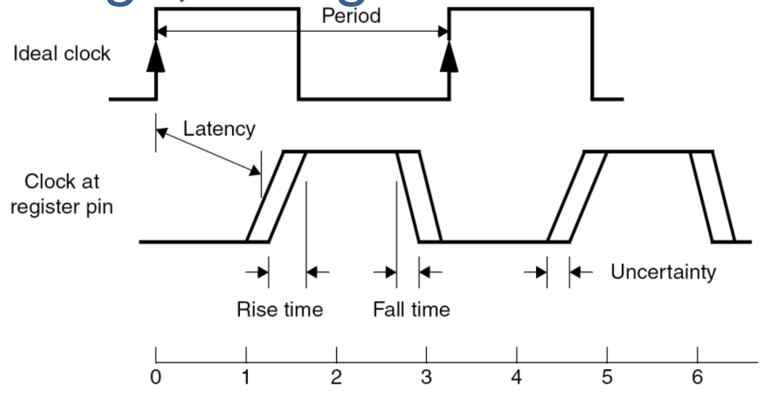
- Creating clock
  - Command:
    - create_clock -name name -period value -waveform {Tr, Tf} port
  - Default clock characteristics (ideal clock)
    - 0 delay at clock port
    - 0 propagation delay
    - 0 transition delay
    - 0 uncertainty
  - Ex:
    - create_clock –period 5.0 –waveform {1.0, 2.0} [get_ports CLK2]



Clock network effects

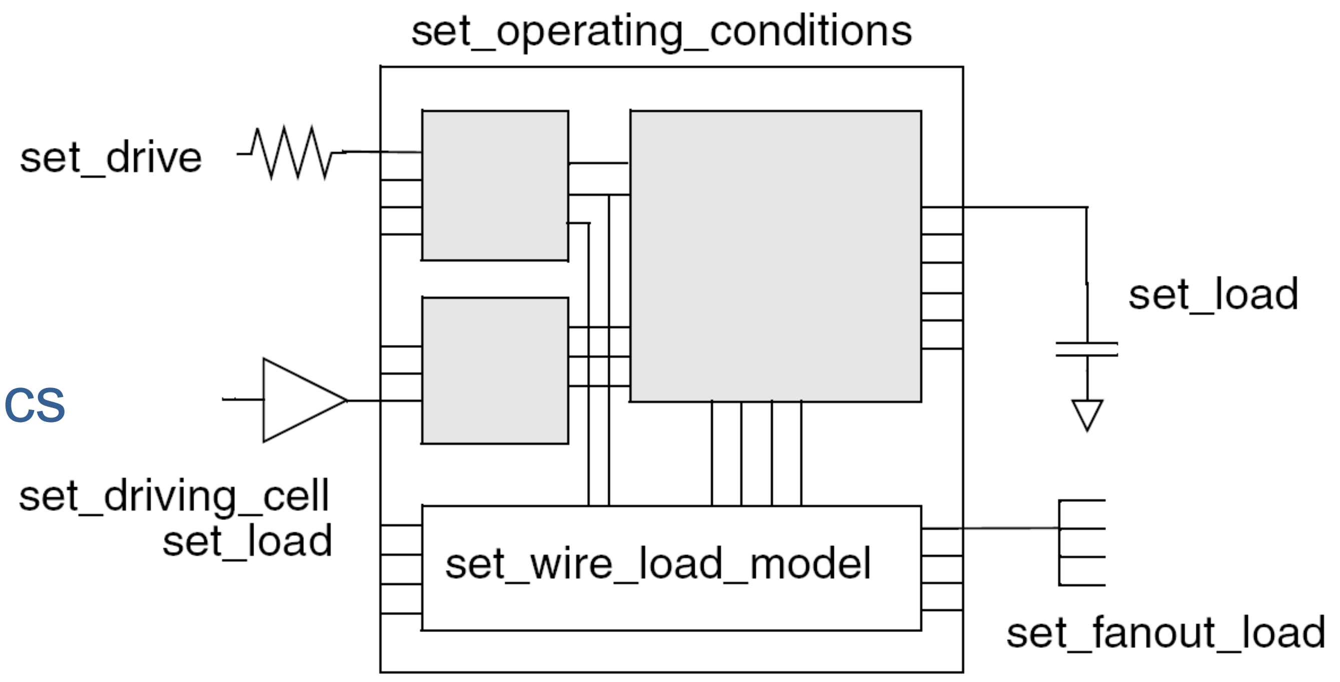


- Clock latency
  - consists of
    - Source latency: clock source to clock pin
    - Network latency: clock pin to register clock pin
- Clock uncertainty (skew)
  - Maximum differences between the arrival of clock signals at registers
- Transition time
  - It takes for a signal to change from low to high, or high to low.
  - Transition time of input affects
    - Delay to output
    - Transition time of output



4. Define design environment

- Design environment
  - Operating condition
    - temperature
    - supply voltage
    - manufacturing process
  - Wire load model
  - System interface characteristics
    - input drive
    - input/output load
    - fanout load



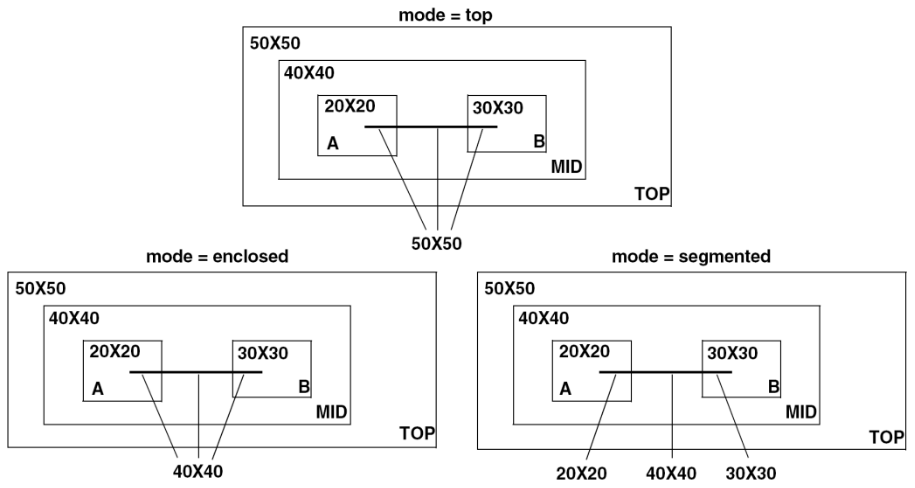
Operating condition

- General operating conditions in technology process
  - best case (fast)
  - typical case
  - worst case (slow)
- List available operation condition (library should be link before.)
  - Command
    - read_file my_lib.db
    - report_lib my_lib
- Specify operating condition
  - Command
    - set_operating_conditions op_name -library library_name
    - set_operating_conditions -max wost_op -min best_op

```
set WORST_OP "slow"
set BEST_OP "fast‟
…
if [array exist BEST_LIB] {
  set_operating_conditions -max $WORST_OP -min $BEST_OP
} else {
  set_operating_conditions -max $WORST_OP
}
```

Wire load model

- Wire load model is defined in library.
- Wire effect estimation
  - wire length
  - fanout
- 3 Hierarchical wire load modes
  - Top (most rough estimation)
    - no subdesigns' wire load models
  - Enclosed
    - depends on its level
  - Segmented (most accurate estimation)
    - depends on its positions.



5. Set design constraints

- Two types of constraints in DC
  - Design rule constraints (precedence)
    - include
      - max. transition time
      - max. fanout
      - max/min. capacitance
    - are defined by technology library.
    - can be defined more restrictively by designer.
  - Optimization constraints
    - include
      - area
      - speed
    - are defined by designer.

Design rule constraints

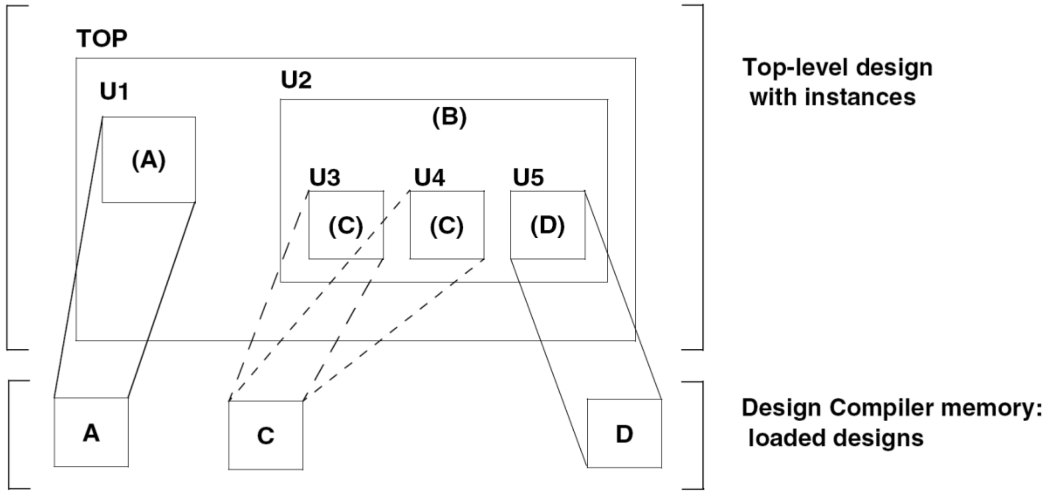
- Max transition time (ns)
  - is the longest time to drive a pin and change its value.
  - command: set_max_transition time IN/OUT/design
- Max fanout (no unit)
  - ∑fanout_load <= max_fanout
  - A high drive strength cell has higher fanout.
  - command: set_max_fanout value IN/design
- Max capacitance
  - is total capacitive load(net) that an output pin can drive.
  - command: set_max_capacitance C IN/OUT/design
- Min capacitance
  - specifies min load that a cell drive.
  - command: set_min_capacitance C IN
- Typical design rule scenarios
  - set_max_fanout + set_max_transition
  - set_max_fanout + set_max_capacitance

Optimization constraints

- Timing constraint
  - sets input delay on pins or input/output ports relative to a clock signal.
  - command: set_input_delay delay –clock clock in
  - command: set_output_delay delay –clock clock out
- Max area
  - unit: number of gates, instead of physical area
  - ignored components in optimization
    - unknown components
    - components with unknown area
    - technology-independent generic cells
  - command: set_max_area area
  - area = 0 (as small as possible)

6. Select compile strategy

- Three compile strategies
  - Top-down compile
  - Bottom-up compile
  - Mixed compile
- Methods for resolving instances
  - Uniquify method
  - Compile-once-don't-touch method
  - Ungroup method


TOP
U1
(A)
U2
(B)
U3
U4
U5
(C)
(C)
(D)
Top-level design with instances
A
C
D
Design Compiler memory: loaded designs

7. Optimization

- DC performs 3 optimization levels
  - Architectural optimization on HDL descrition
    - Sharing common expressions
    - Sharing resources
    - Selecting DesignWare implementations
    - Reordering operators
    - Identifying arithmetic expressions for data-path synthesis (DC-ultra)
  - Logic-level optimization on GTECH netlist
    - Structuring
    - Flattening for conditional logic
  - Gate-level optimization on technology-dependent netlist
    - Technology mapping
    - Delay optimization
    - Design rule fixing
    - Area optimization

compile

- Default synthesis algorithm
  - command
    - compile
    - compile –map_effort median –area_effort median
- Advanced synthesis algorithms
  - High-performance designs
    - compile_ultra
  - Maximum performance
  - Minimum area
  - Data path
- More information
  - Refer to “Design Compiler Optimization Reference Manual”

7. Analyze design problems

Object	Command	Description
Design	`report_design`	Reports design characteristics.
	`report_area`	Reports design size and object counts.
	`report_hierarchy`	Reports design hierarchy.
	`report_resources`	Reports resource implementations.
Instances	`report_cell`	Displays information about instances.
References	`report_reference`	Displays information about references.
Pins	`report_transitive_fanin`	Reports fanin logic.
	`report_transitive_fanout`	Reports fanout logic.
Ports	`report_port`	Displays information about ports.
	`report_bus`	Displays information about bused ports.
	`report_transitive_fanin`	Reports fanin logic.
	`report_transitive_fanout`	Reports fanout logic.
Nets	`report_net`	Reports net characteristics.
	`report_bus`	Reports bused net characteristics.
	`report_transitive_fanin`	Reports fanin logic.
	`report_transitive_fanout`	Reports fanout logic.
Clocks	`report_clock`	Displays information about clocks.

Area

- Analyzing area
  - Combinational area
  - Non-combinational area
  - Total area
- Report for current design
  - report_area
- Report for current design across the hierarchy
  - report_area -heirarchy

```
****************************************
Report : area
Design : JBF
Version: B-2008.09-SP2
Date   : Wed Dec 23 20:38:36 2009
****************************************

Library(s) Used:
……

Number of ports:                  169
Number of nets:                 70269
Number of cells:                55211
Number of references:             205

Combinational area:        669475.743010
Noncombinational area:     1593732.288294
Net Interconnect area:     13561542.382202

Total cell area:           2263208.031303
Total area:                15824750.413506
```

Timing

- report_design
  - Shows operating conditions, wire load model and mode, timing ranges, internal input and output, and disabled timing arcs.
- check_timing
  - Checks for unconstrained timing paths and clock-gating logic.
- report_port
  - Shows unconstrained input and output ports and port loading.
- report_timing_requirements
  - Shows all timing exceptions set on the design.
- report_clock
  - Checks the clock definition and clock skew information.
- report_path_group
  - Shows all timing path groups in the design.
- report_timing
  - Checks the timing of the design.
  - Slack should be 0 or positive.
- report_constraint
  - Checks the design constraints.
- report_delay_calculation
  - Reports the details of a delay arc calculation.

8. Change naming rule

- Goal
  - Change naming rule in netlist to avoid confliction in APR.
- Variables
  - bus_inferface_sytle
    - individual bits of bus
  - bus_naming_style
    - individual port member, net member, or cell instance
  - hdlout_internal_busses
    - internal bused nets by parsing the names of the nets.

```
set bus_inference_style {%s[%d]}
set bus_naming_style {%s[%d]}
set hdlout_internal_busses true
```

9. Save design

- database (.db)
  - Command: write –hierarchy –format db –output file.db
- netlist (.v)
  - IEEE standard Verilog
  - Command: write –hierarchy –format verilog –output file.v
- standard delay format (.sdf)
  - Delay model for gate-level simulation
  - Command: write_sdf –context verilog –version 1.0 file.sdf
- standard parasitic extraction format(.spef)
  - Netlist with RC information
  - Command: write_parasitics –format reduced –output file.spef
  - -format reduced: one resistance and one capacitance for net model
- Synopsys design constraints (.sdc)
  - Command: write_sdc file.sdc

Gate-level simulation

- Command
  - ncverilog testbench –v tech_model.v +access+r
  - tech_model.v
    - Path: /CAD/UMC0090/UMC90_CELL/UMC90_CELL/SP_RVT_C05/verilog
    - File: l90sprvt.v