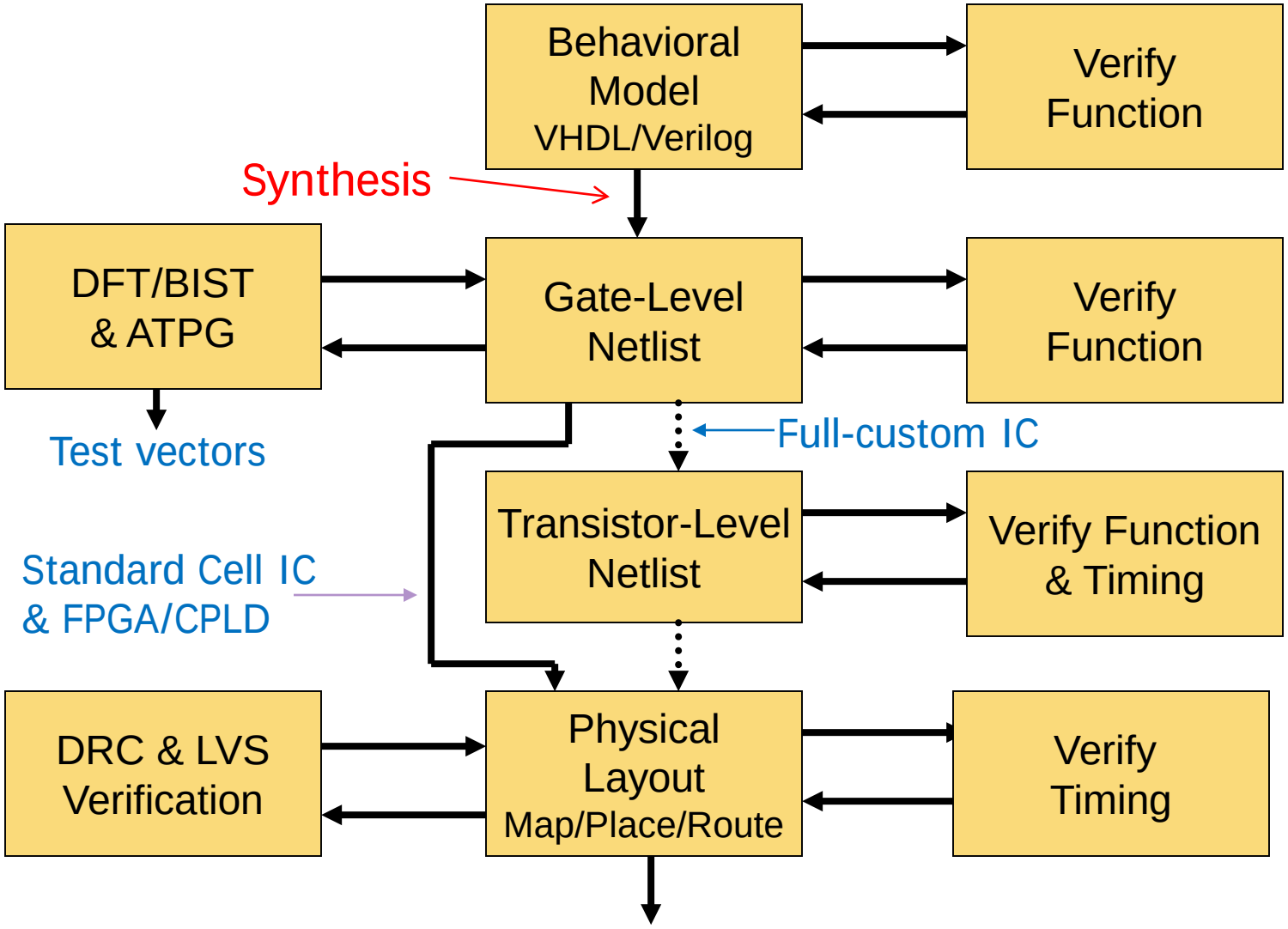
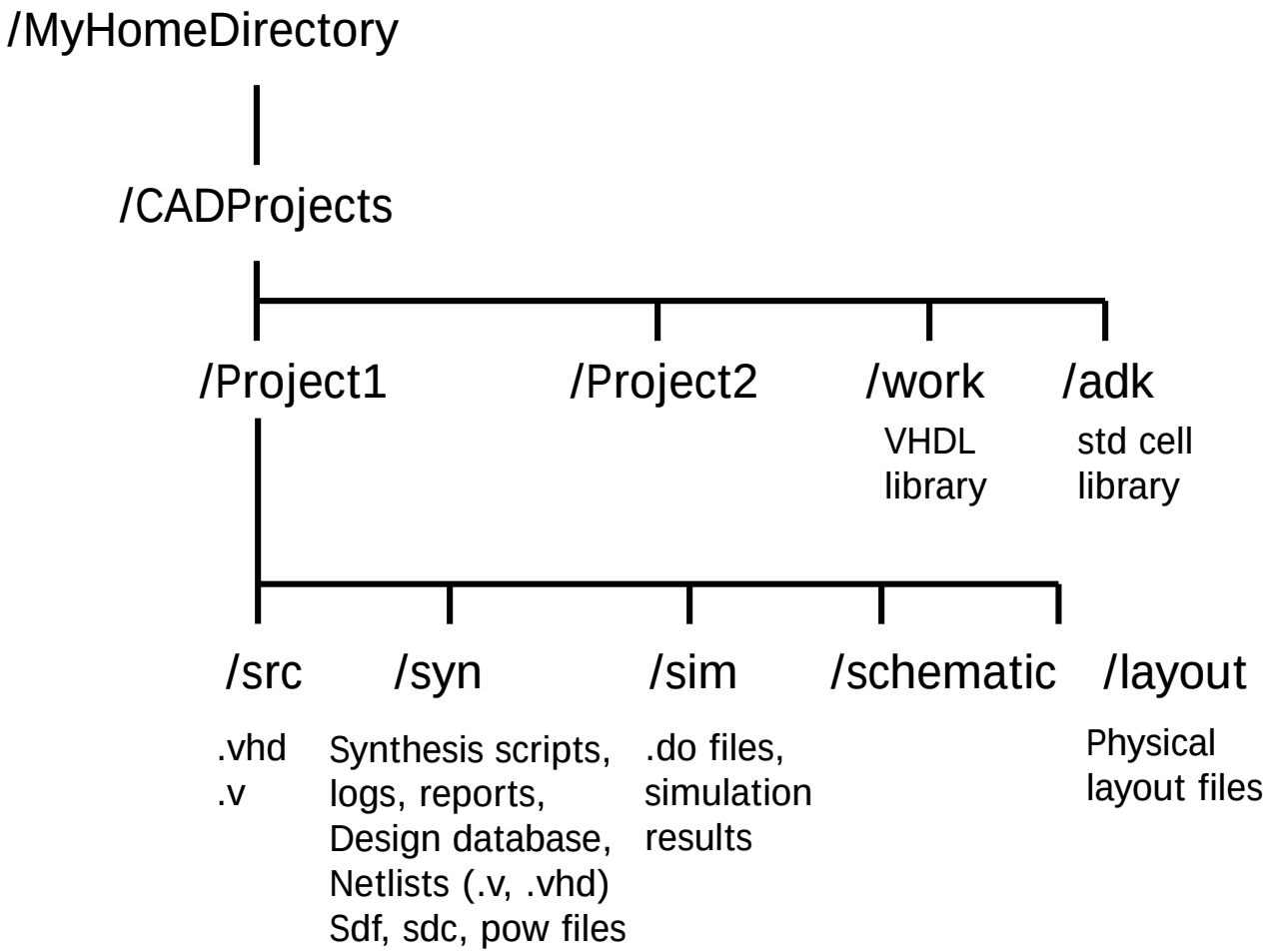
Automated Synthesis from HDL models

Leonardo (Mentor Graphics), Design Compiler (Synopsys)

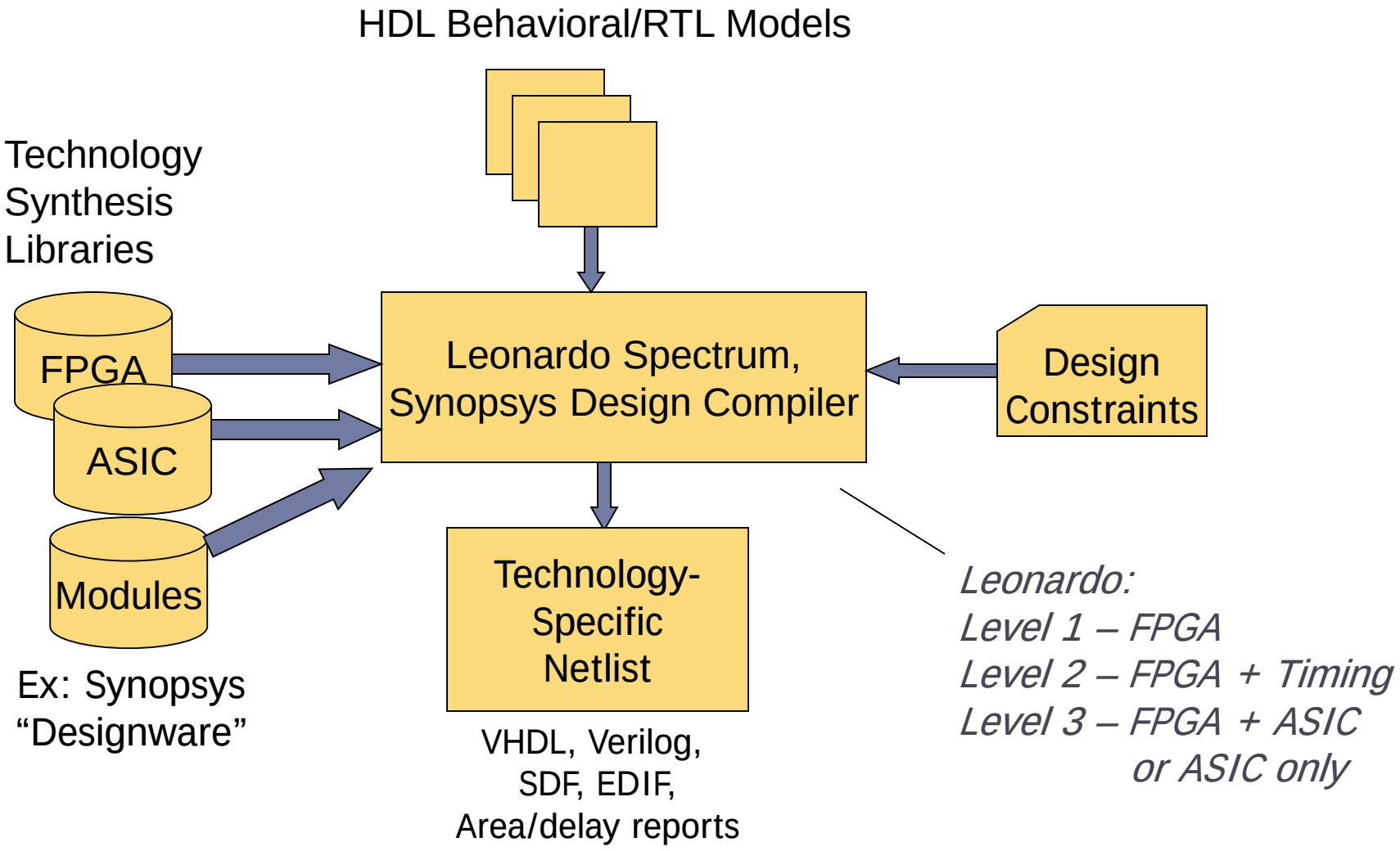
ASIC Design Flow



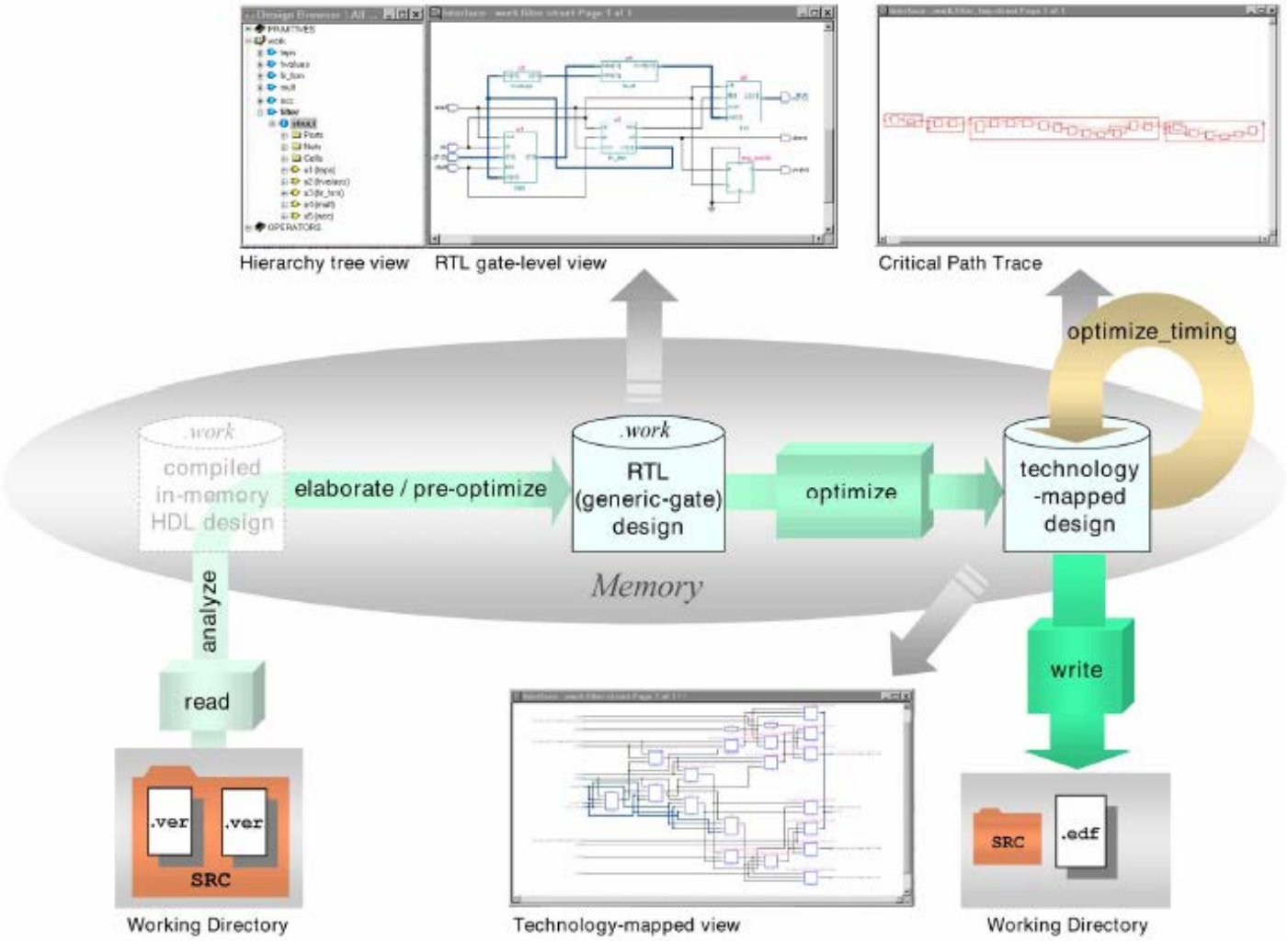
Project directory structure



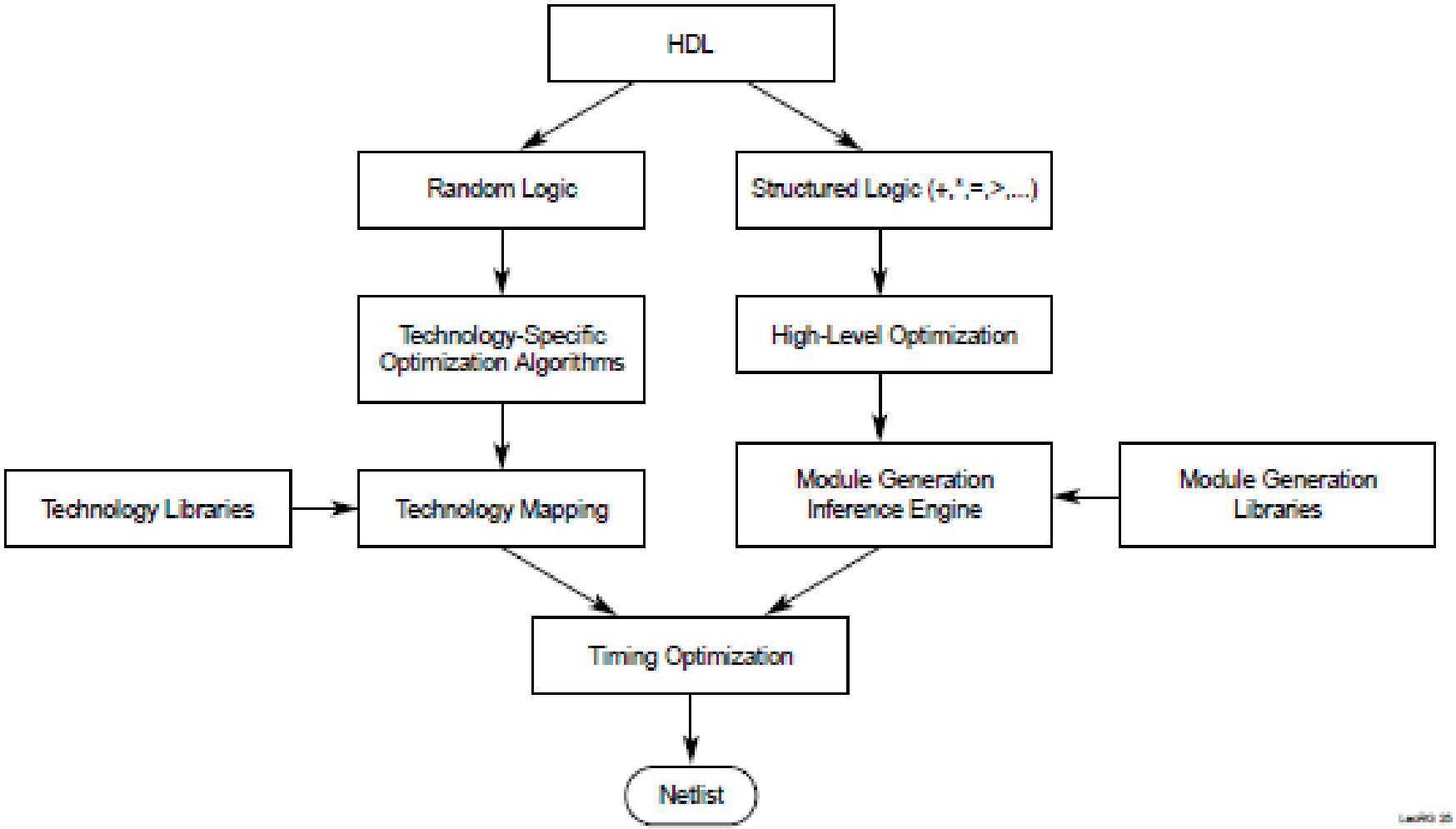
Automated synthesis



Leonardo – ASIC synthesis flow



Synthesis design flow



Synthesis steps

1. Load technology library into database
2. Analyze design
   - Load HDL models into database, check for synthesizable models
3. Elaborate design
   - Technology-independent circuit (random & structured logic)
4. Specify design constraints (timing, area)
5. Compile/optimize design
   - Optimize for the loaded technology library
   - Repeat as necessary to meet constraints
6. Generate technology-specific netlist(s)
7. Generate simulation timing data (SDF file)
8. Generate reports (cells, area, timing)

LeonardoSpectrum Documentation

To access documentation (Linux):

- In .bashrc file:

```
export EXEMPLAR=/linux_apps/mentor/LeonardoSpectrum/2014a
export LEO_DOCS=$EXEMPLAR/doc/_bk_leospec.pdf
```

- From command line:

```
mgcdocs $LEO_DOCS
```

- In GUI – access documents from the help menu

Main Documents:

- User's manual
- Reference Manual (Command summaries)
- HDL Synthesis Manual (VHDL/Verilog for synthesis)
- Synthesis and Technology Manual (ASIC/FPGA-specific)

Invoking LeonardoSpectrum

1. Invoke command line version (Linux): *spectrum*
   - "Spectrum" is command-line version for Linux
   - To execute command file after startup:

     spectrum –file mycommands.tcl

2. To invoke GUI version: *leonardo*
   - Execute most commands interactively
   - Can save "transcript" to use as basis for command file

LeonardoSpectrum synthesis example

- Load technology library: tsmc035 (ASIC)
- Load design file: modulo7.vhd
- Specify constraints: clock freq, delays, etc.
- Optimization: effort, performance vs. area
- Write synthesized netlist output(s):
  - modulo7_0.vhd :VHDL netlist for ModelSim & DFT
  - modulo7.v :Verilog netlist for import into DA-IC
  - modulo7.sdf : For ModelSim to study timing
  - modulo7.edf : EDIF netlist for 3rd party tool (Xilinx)

Behavioral model to be synthesized

```
-- modulo-7 counter with asynchronous reset and synchronous load/count
library ieee;  use ieee.std_logic_1164.all;  use ieee.numeric_std.all;
entity modulo7 is
port( count, load, reset, clk:  in std_logic;
        I:  in    unsigned(2 downto 0);        -- "unsigned" form of std_logic_vector
        Q: out unsigned(2 downto 0));      -- defined in IEEE "numeric_std" package
end modulo7;
architecture Behave of modulo7 is
     signal Q_s: unsigned(2 downto 0);
begin
     process (reset,clk) begin
          if (reset='0') then Q_s <= "000";    -- async reset
          elsif (clk'event and (clk='1')) then
                if (count = '1') and (Q_s = "110") then Q_s <= "000";   -- count rolls over
                elsif (count='1') then  Q_s <= Q_s + 1;   -- increment count
                elsif (load='1') then  Q_s <= I;  -- synchronous load
                end if;
          end if;
     end process;
     Q<=Q_s;  -- drive the outputs
end;
```

Synthesized netlist (1)

```
-- Definition of  modulo7
--      Thu Sep 21 10:48:09 2006
--      LeonardoSpectrum Level 3, 2005a.82
--
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity modulo7 is
   port (
      count : IN std_logic ;
      load : IN std_logic ;
      reset : IN std_logic ;
      clk : IN std_logic ;
      I : IN std_logic_vector (2 DOWNTO 0) ;
      Q : OUT std_logic_vector (2 DOWNTO 0)) ;
end modulo7 ;
```

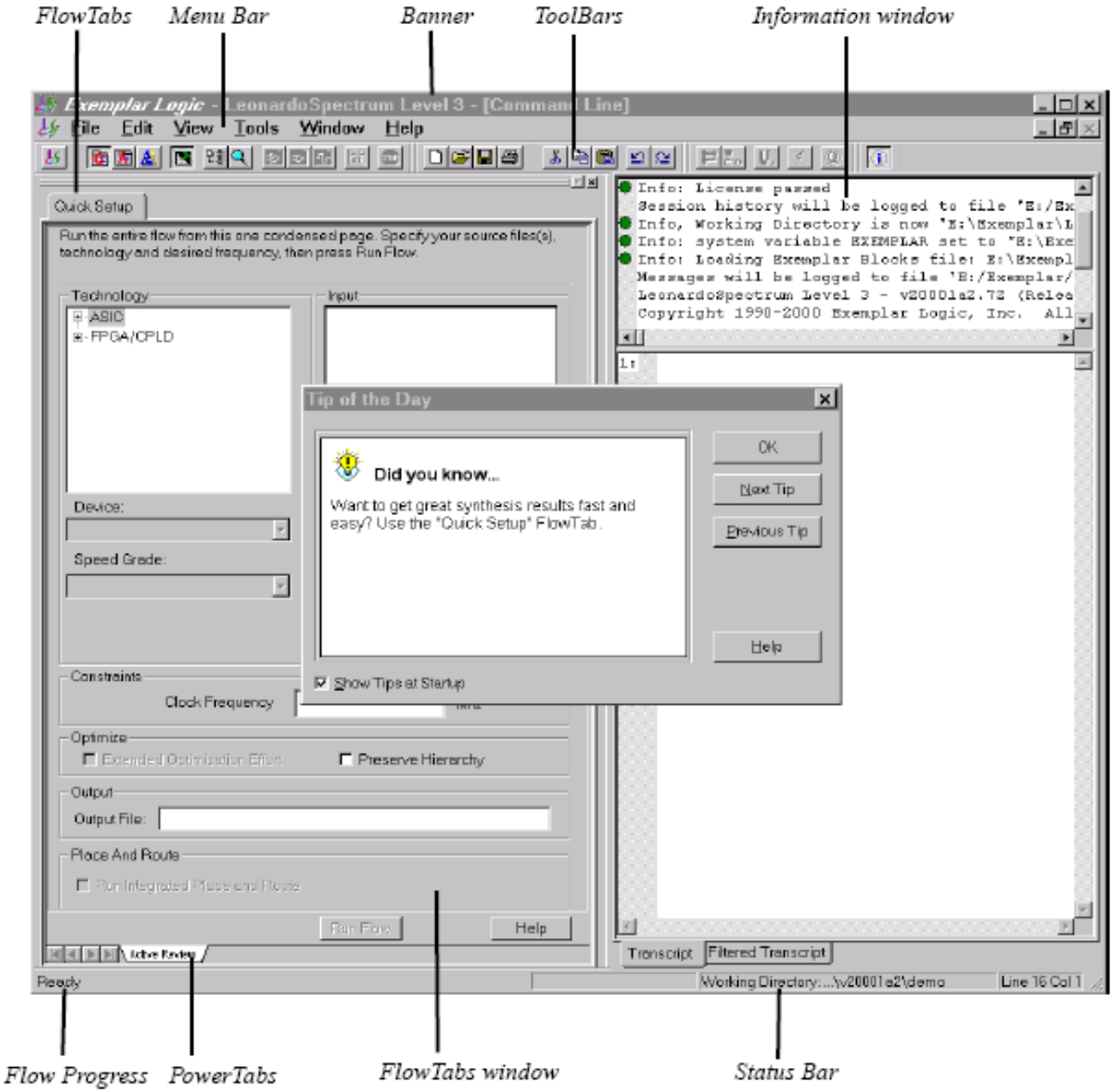
Synthesized netlist (2)

```
architecture Behave of modulo7 is
   signal Q_2_EXMPLR, Q_1_EXMPLR, Q_0_EXMPLR, NOT_reset, nx4, nx14, nx22,
      nx48, nx60, nx169, nx179, nx189, nx202, nx204, nx208, nx212, nx214,
      nx218, nx225, nx228, nx230: std_logic ;
begin
   Q(2) <= Q_2_EXMPLR ;
   Q(1) <= Q_1_EXMPLR ;
   Q(0) <= Q_0_EXMPLR ;
   ix170 : mux21_ni port map ( Y=>nx169, A0=>nx14, A1=>Q_0_EXMPLR, S0=>nx225);
   ix15 : oai22 port map ( Y=>nx14, A0=>Q_0_EXMPLR, A1=>nx202, B0=>nx230,
                           B1=>count);
   ix203 : nand02 port map ( Y=>nx202, A0=>count, A1=>nx204);
   ix205 : nand04 port map ( Y=>nx204, A0=>count, A1=>Q_2_EXMPLR,
                             A2=>Q_1_EXMPLR, A3=>nx228);
   ix180 : oai32 port map ( Y=>nx179, A0=>nx208, A1=>count, A2=>load,
                            B0=>nx212, B1=>nx225);
   Q_2_EXMPLR_EXMPLR : dffr port map ( Q=>Q_2_EXMPLR, QB=>nx208, D=>nx179,
                                       CLK=>clk, R=>NOT_reset);
```

Synthesized netlist (3)

```
  ix211 : inv01 port map ( Y=>NOT_reset, A=>reset);
  ix213 : aoi22 port map ( Y=>nx212, A0=>I(2), A1=>nx214, B0=>nx22, B1=>nx4 );
  ix216 : inv01 port map ( Y=>nx214, A=>count);
  ix219 : nand02 port map ( Y=>nx218, A0=>Q_1_EXMPLR, A1=>Q_0_EXMPLR);
  Q_1_EXMPLR_EXMPLR : dffr port map ( Q=>Q_1_EXMPLR, QB=>OPEN, D=>nx189,
                                      CLK=>clk, R=>NOT_reset);
  ix190 : mux21_ni port map ( Y=>nx189, A0=>nx60, A1=>Q_1_EXMPLR, S0=>nx225);
  ix61 : ao32 port map ( Y=>nx60, A0=>nx48, A1=>nx218, A2=>nx4, B0=>I(1),
                         B1=>nx214);
  ix49 : or02 port map ( Y=>nx48, A0=>Q_0_EXMPLR, A1=>Q_1_EXMPLR);
  ix226 : nor02_2x port map ( Y=>nx225, A0=>count, A1=>load);
  Q_0_EXMPLR_EXMPLR : dffr port map ( Q=>Q_0_EXMPLR, QB=>nx228, D=>nx169,
                                      CLK=>clk, R=>NOT_reset);
  ix231 : inv01 port map ( Y=>nx230, A=>I(0));
  ix5 : inv01 port map ( Y=>nx4, A=>nx202);
  ix23 : xor2 port map ( Y=>nx22, A0=>nx208, A1=>nx218);
end Behave ;
```

Leonardo Main Window



Leonardo
Quick Setup Tab

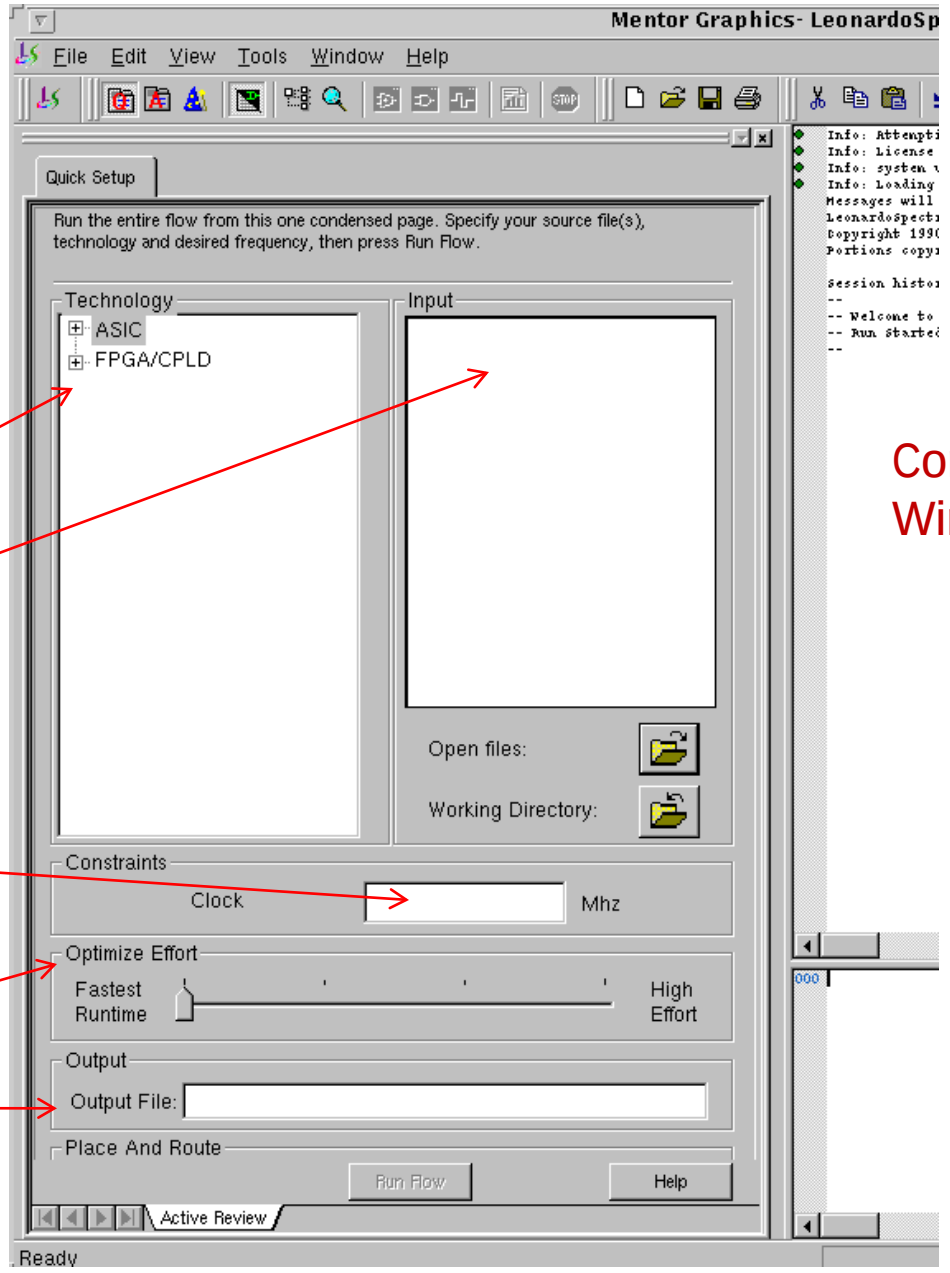
Technology

HDL Files

Target Clock Freq

Optimization Effort

Output File



Command Window

Spectrum commands/attributes/variables

- Enter "commands" at the command prompt
  - More efficient to read commands from a Tcl script file.
- A "variable" specifies a global constraint, directive, etc.
  - Tcl "set" and "unset" commands change variables

```
set voltage 3.3
```

- An "attribute" is information attached to an object in the memory design database.
  - Allows user to fine-tune the process at the object level.
  - Set with Tcl "set_attribute" command (also "unset_attribute)
  - An attribute has: owner, name, value
  - Example:

```
set_attribute –net n1 –name max_fanout_load –value 10
```

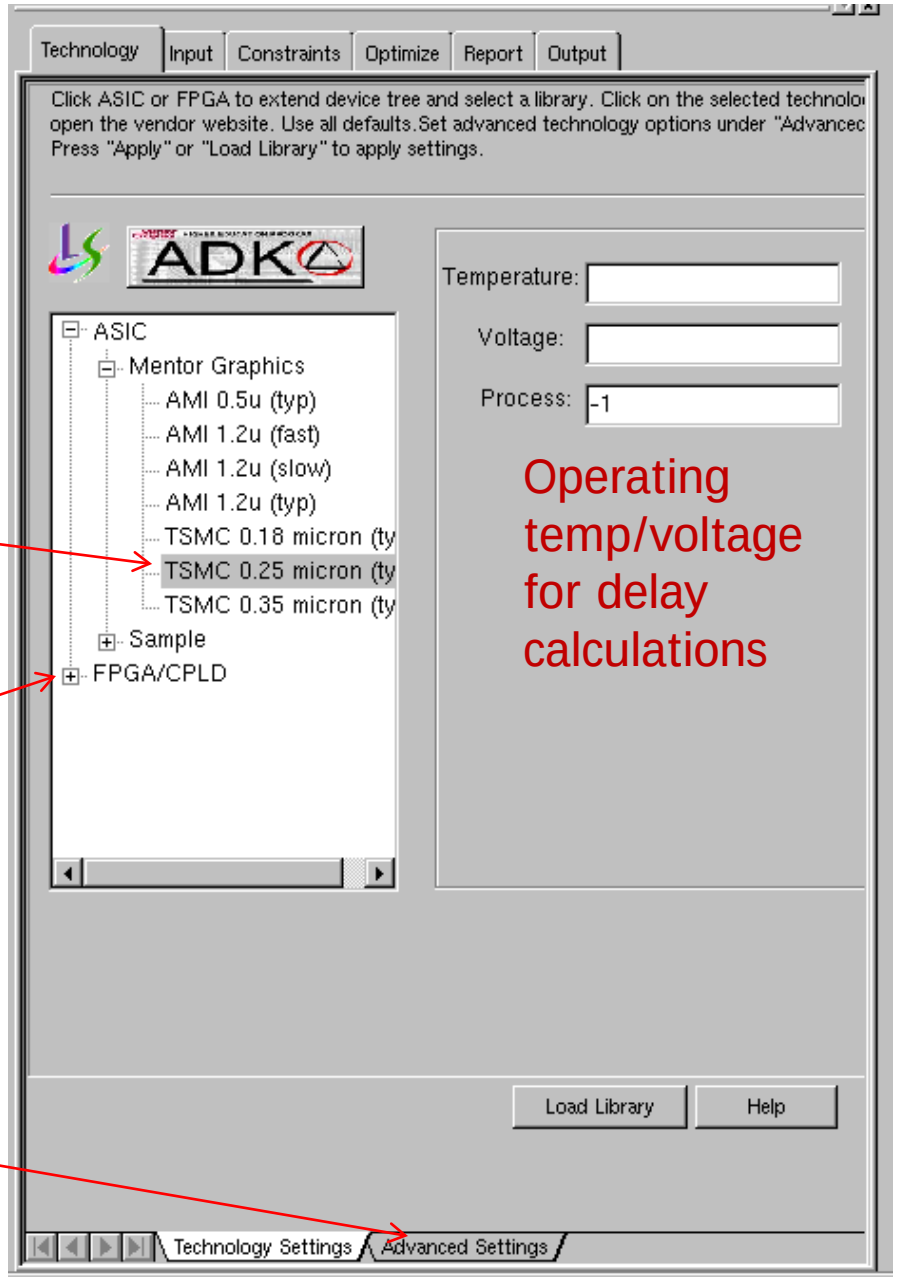
Technology library (ASIC)

- Load synthesis library for target technology into memory
  - Command example: load_library technology-name
    - Default location: $EXEMPLAR/lib/technology-name.syn
- Command to load ADK TSMC 0.35µm (350nm) library:
  - *load_library $EXEMPLAR/lib/tsmc035_typ*
- Available ADK libraries:
  - tsmc035_typ.syn – use for course projects
  - tsmc025_typ.syn
  - tsmc018_typ.syn
  - ami12_typ.syn
  - ami05_typ.syn
  - gdk.syn

Technology library (ASIC) variables

- Technology variables affect delay calculations ("delay derating factors")
  - Use tech library defaults if variables not set
  - set voltage 2.5 (volts)
  - set temp 40 (degrees celsius/centigrade)
  - set process 1 (process variation # – if available)

Technology Tab (ASIC)



Choose ASIC library

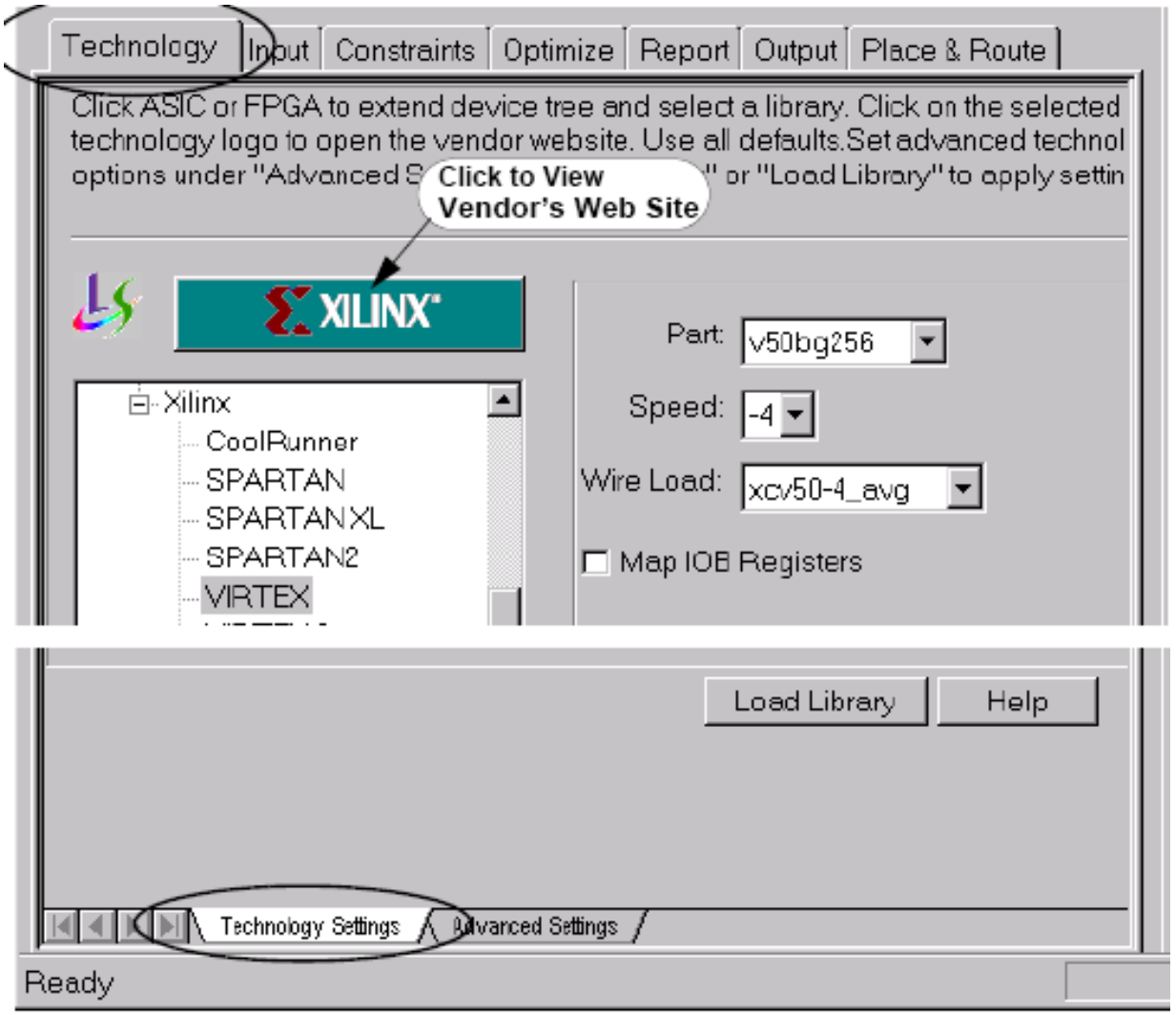
FPGA options include device family, part, speed grade and pin selection

Operating temp/voltage for delay calculations

Advanced settings: Select max fanout

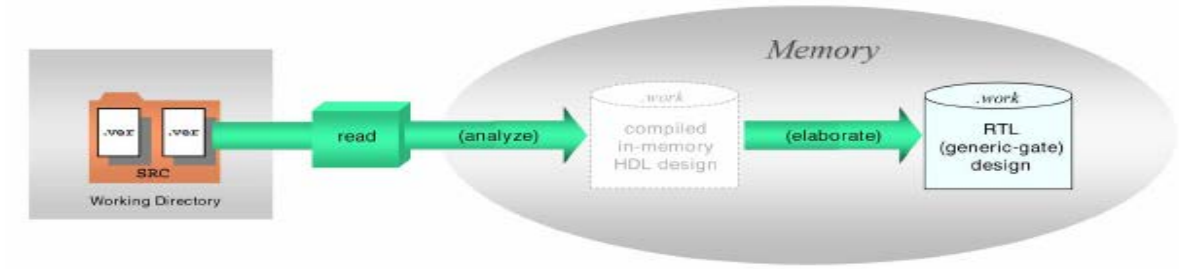
Technology Tab (FPGA)

Specify:
- part,
- speed grade,
- wire load,
- use of IOB registers



Load a design into the database

- Analyze – syntax check and build database
  - Input VHDL and/or Verilog models
  - check dependencies & resolve generics/parameters
- Elaborate – synthesize to generic gates and black boxes
  - technology-independent gates
  - operators (arithmetic, relational, etc.) recognized and implemented with "black boxes" (no logic in them yet)



- Read command does analyze + elaborate + pre-optimize

Analyze Command

- analyze {f1.vhd src/f2.vhd "top file.vhd"}
  - Read and analyze into default memory database library "work"
  - List VHDL files in bottom-up order – top level last
  - Use quotes if embedded spaces in file name: "top file.vhd"
  - Include directory if necessary: src/f2.vhd
- Analyze command switches:
  - -format vhdl (or verilog) [default VHDL if file ext = .vhd/.vhdl or Verilog if file ext = .v/.verilog]
  - -work lib_name [lib where design to be stored (default = "work".) Different libraries might be used for comparing designs]
- Examples:
  - analyze {src/f1.vhd src/f2.vhd}
  - analyze {src/f1.vhd src/f2.vhd} –work lib_version1

Elaborate Command

- "Elaborate" a design currently in the memory database – producing tech-independent circuit
  - elaborate divider ["divider" = VHDL entity/Verilog module]
  - Switches
    - -single_level [only do top level – for bottom-up design]
    - -architecture a1 [if other than most recently analyzed]
    - -work lib_name [if name other than work]
    - -generics { size=9 use_this=TRUE initval="10011" }
      - List format is { generic=value generic=value …. }
    - -parameters [format same as generics]

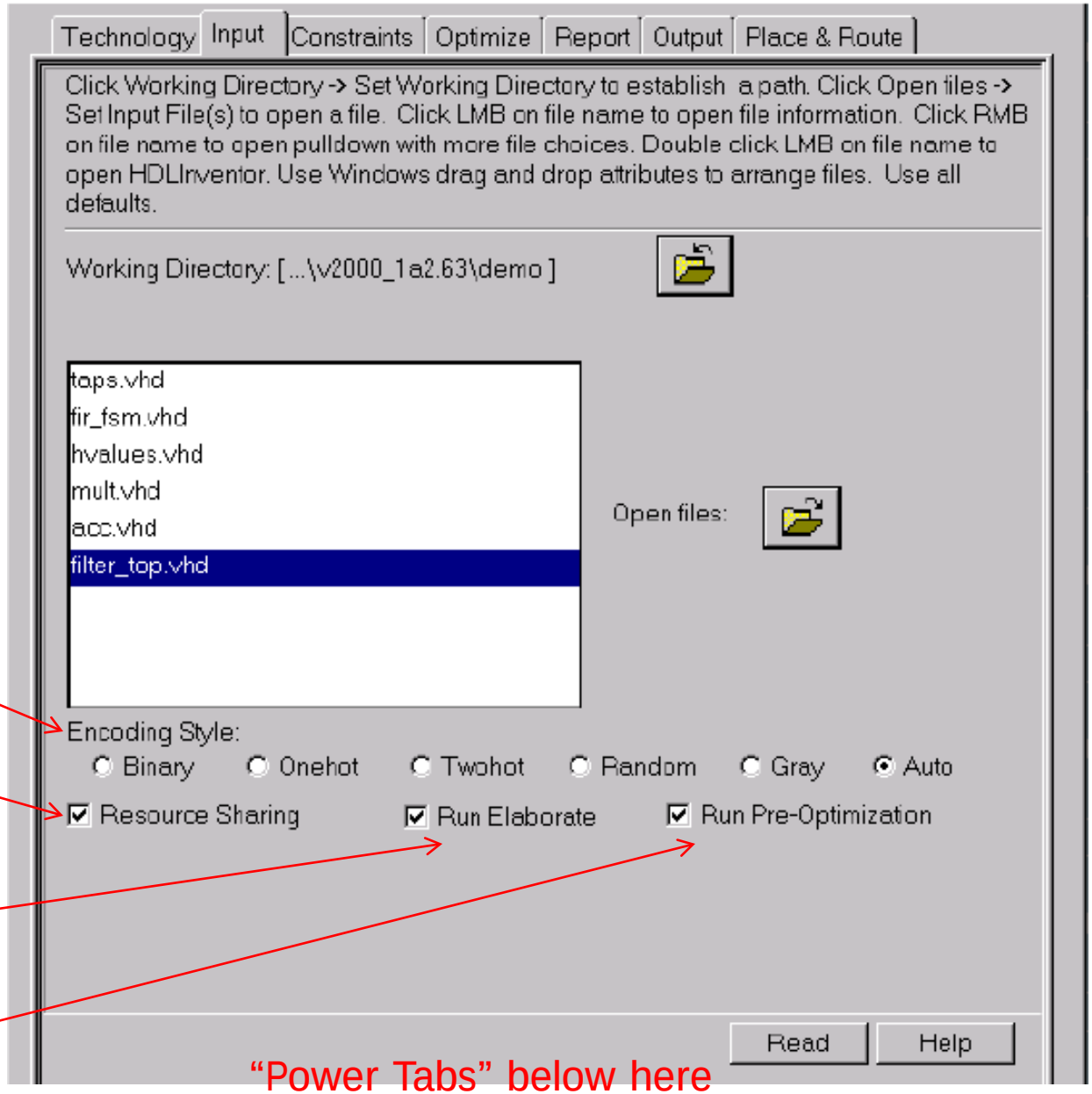
Read command

- Performs both analyze and elaborate steps
  - read {f1.vhd src/f2.vhd "top file.vhd"}
  - Same switches as *analyze* and *elaborate* commands, plus:
    - -dont_elaborate {f1.vhd} – do analysis but not elaborate

- GUI: Input Flow Tab can be used to run read/analyze/elaborate
  - Uncheck "Run Elaborate" box for analyze only
  - Uncheck "Run Pre-Optimization" to skip that step for now
  - Select other analyze/elaborate options via the "Power Tabs"

Input Flow Tab

HDL file list:
List VHDL files from bottom to top levels of hierarchy.



FSM encoding

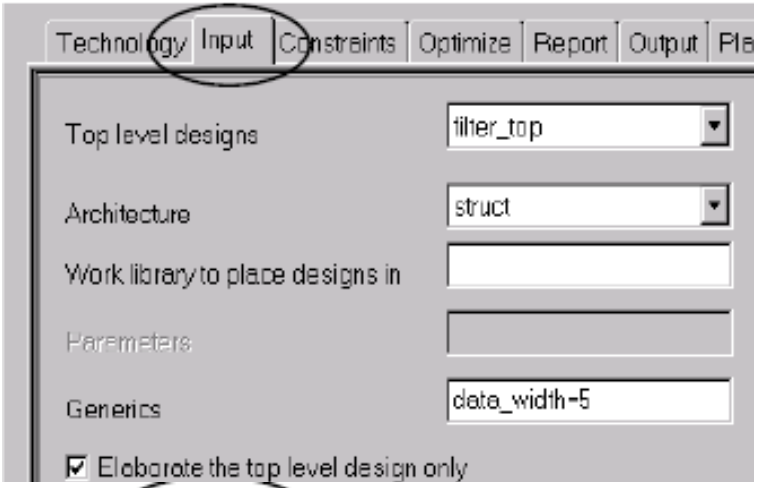
Share resources (adders, etc.) to do multiple op's

Run Elaborate step after Analysis.

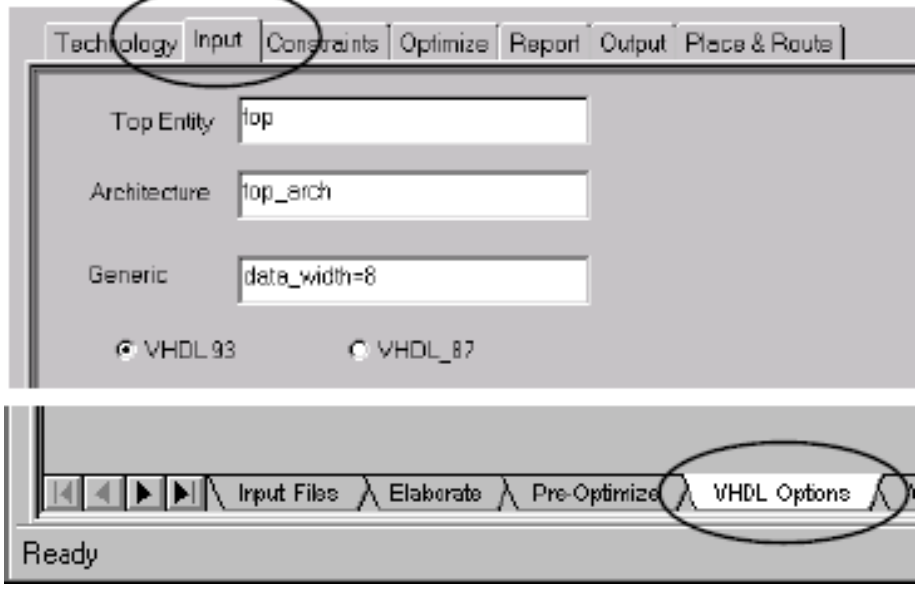
Run Pre-Optimize step after Elaborate

"Power Tabs" below here

Elaborate & VHDL Options Power Tabs



Elaborate: Select:
top-level design name,
architecture,
work library,
generics,
top-level only



VHDL Options:
Select VHDL 87 instead of 93

Other power tabs for other formats (Verilog, SDL, etc.)

Global constraint variables

- Affect design decisions
  - set max_fanout_load 5
    - Global limit on max inputs driven by one output
    - Leonardo splits nets or adds buffers as needed

      (but buffers add delay)
  - Override with max_fanout_load *attribute* on a net:

    set_attribute –net n1 –name max_fanout_load –value 10

    Constrains only net "n1"
  - set max_fanin 5
  - set max_cap_load 4
  - set max_transition 1.2
  - set max_pt 8 (max # product terms in a sum)

Load and Drive Constraint Attributes

- output_load value out_signal1 out_signal2 …

  output_fanout value out_signal1 out_signal2 …
  - # unit loads/fanout loads driven by the output
  - Use to calculate delays/drive capability
  - May need buffers on selected outputs
- input_max_load value in_signal1 in_signal2 …

  input_max_fanout value in_signal1 in_signal2 …
  - Max unit loads/fanout load presented to a circuit input
  - May need inserted buffers to reduce loads
- input_drive value in_signal1 in_signal2 …
  - Additional delay in ns/unit load for an input port

Balancing Loads

- Resolve load violations throughout the design
  - Fix loads after changing attributes, without rerunning optimize
    - Load balancing always done as part of optimize
  - Pays attention to OUTPUT_LOADS, OUTPUT_FANOUTS
- Mostly used at boundaries of hierarchical modules
  - Optimize balances loads within modules
- Command:

  balance_loads [design-name] [-single]

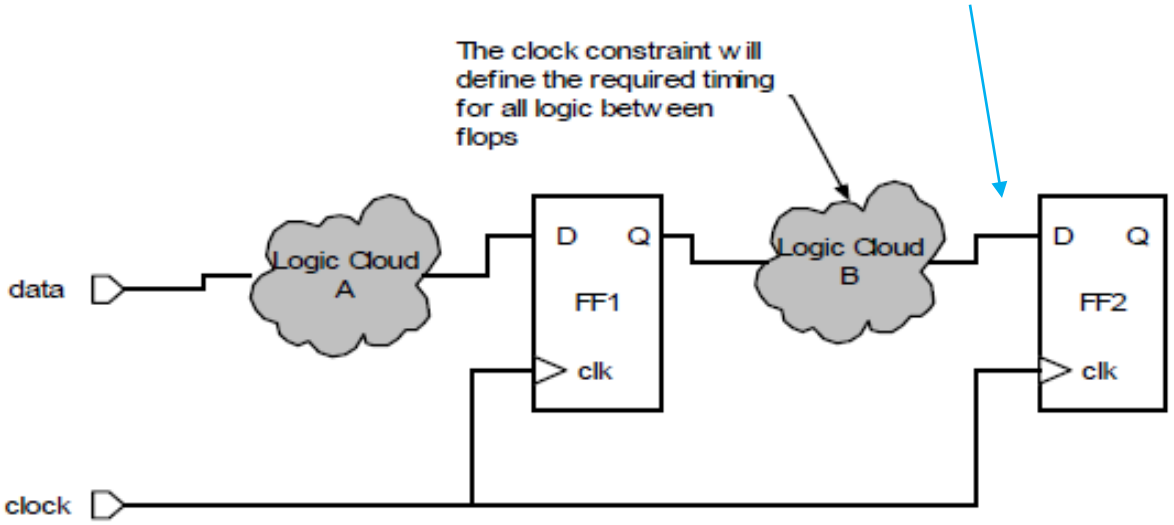
Wire Load Table (not used with ADK/GDK)

- Use to estimate routing delays
  - Exact delays known only after place and route)
  - Function of cell sizes and fanouts
- Table of RC values estimated from net lengths in previous projects (provided by vendor)
- Variables:
  - wire_load_library name
    (lib to which designed mapped - or NIL)
  - wire_table name (if named table loaded)
  - wire_tree (best,balanced,worst, or not set)
  - wire_load_mode (top, segmented)

Timing Constraints

- Simple: specify target clock frequency
- Advanced: specify globally or on specific blocks
  - Clock: period/frequency, pulse width, duty cycle
  - Input: arrival time, transition times, driver strength
  - Output: required time, transition times

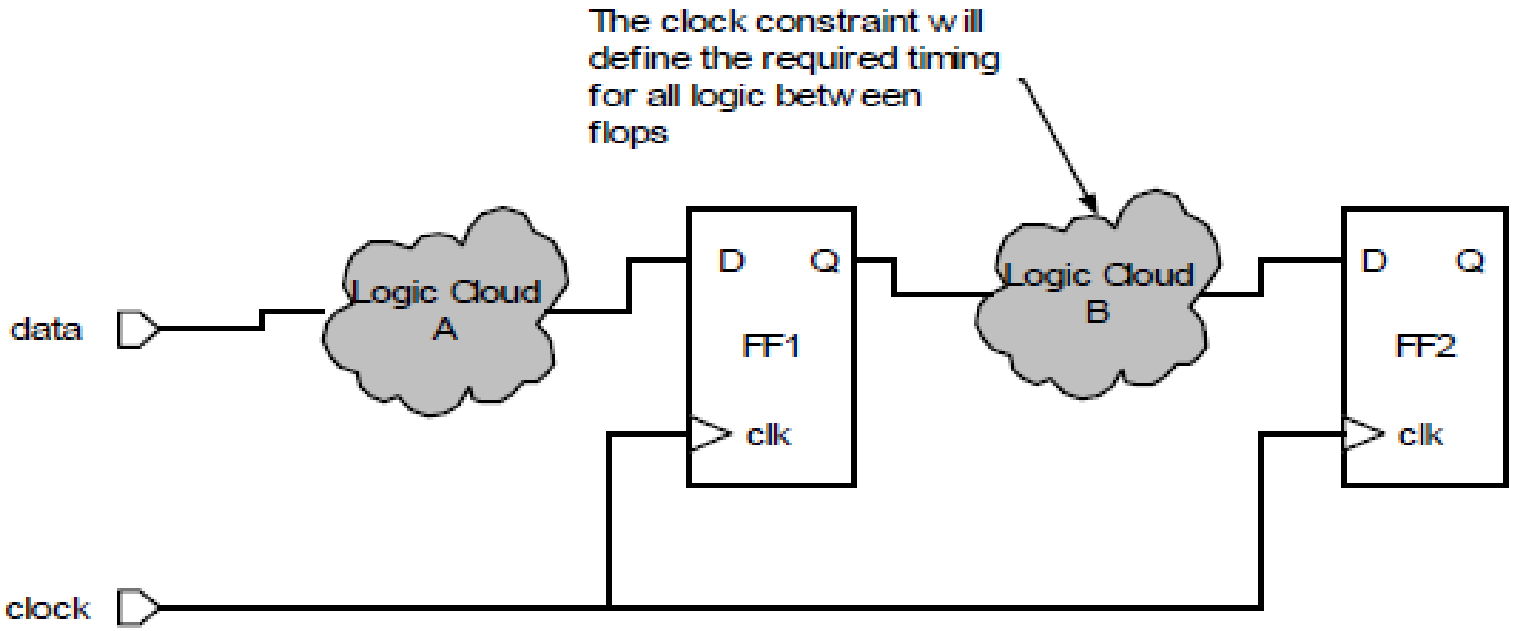
(required time – arrival time = "slack")



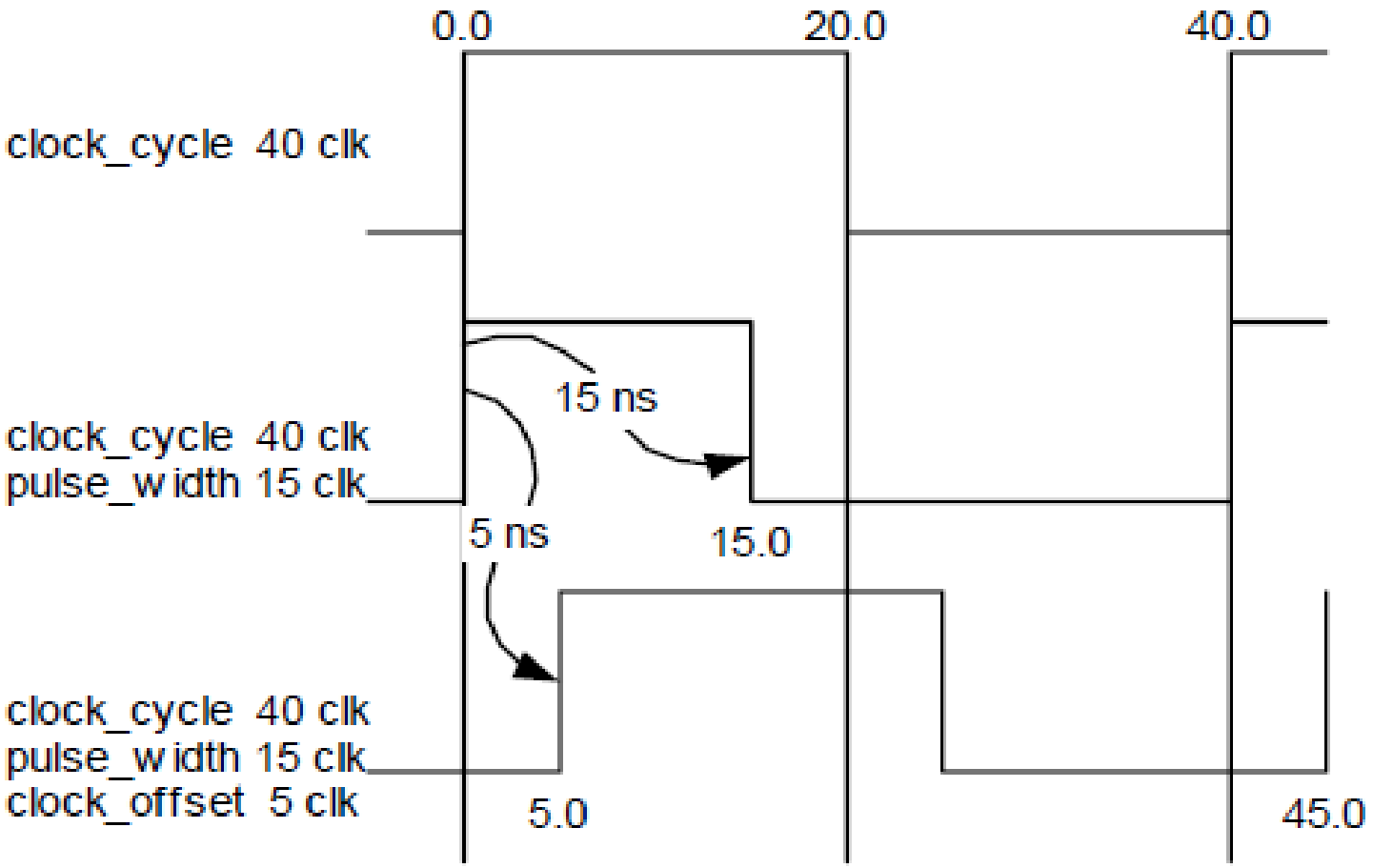
Clock-related constraints

- clock_cycle <clock_period> <primary_input_port>
- pulse_width <clock_pulse_width> <primary_input_port>
- clock_offset <clock_offset> <primary_input_port>

  (offset from time 0 - "skew")



Clock constraint examples



Global

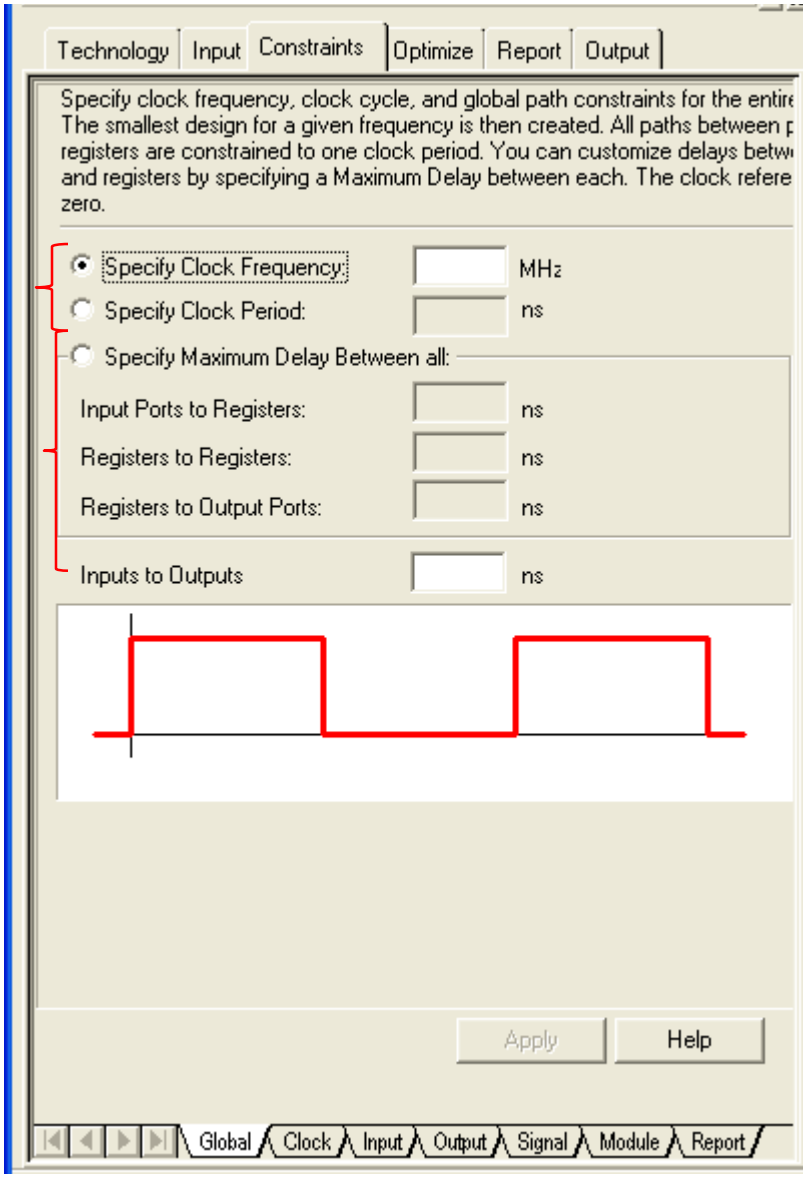
Clock constraints

Frequency/period

Delay constraints

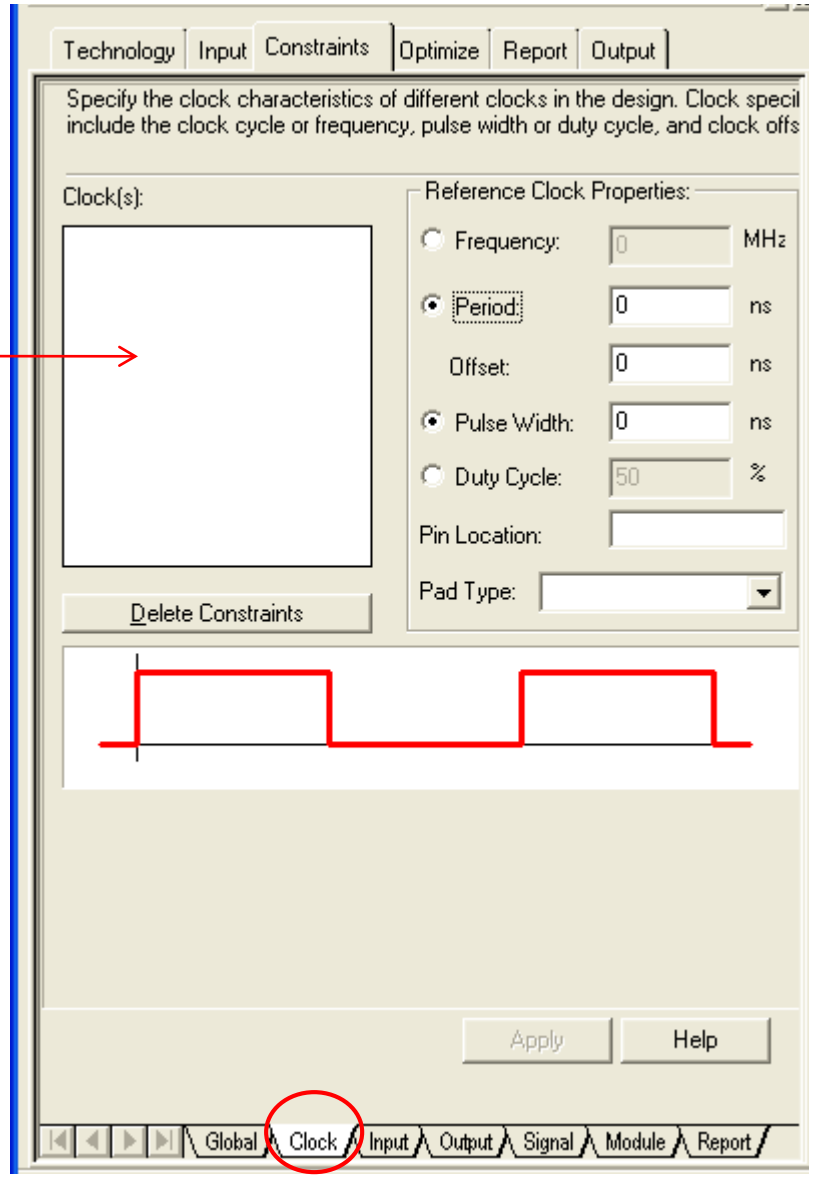
Path delays

These apply to the entire design, except for objects for which attributes have been defined that specify other clock values.



Individual
Clock net attributes

Clock name(s)



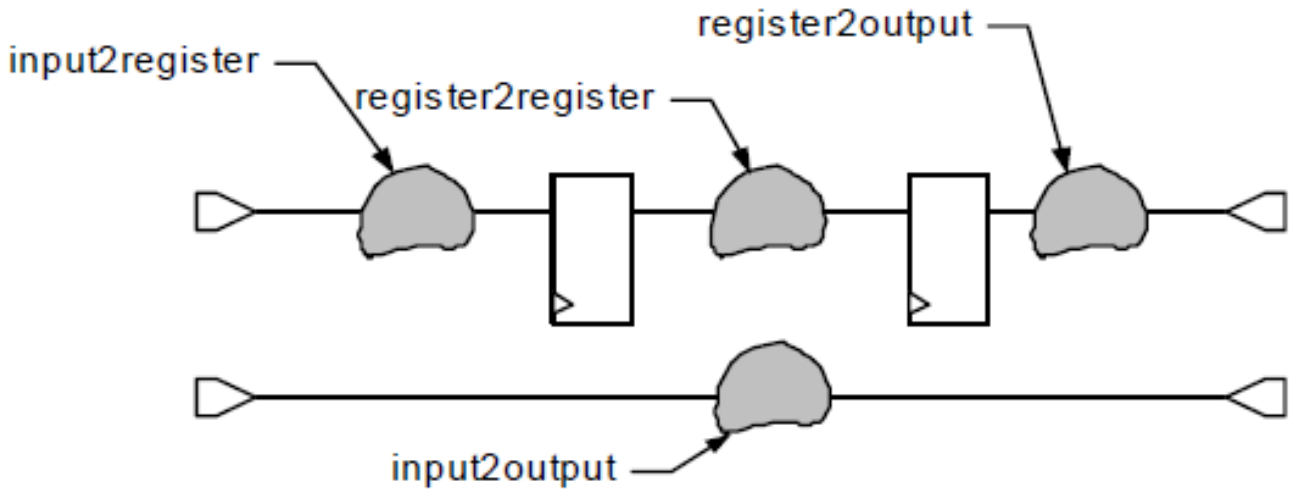
These override the global clock settings for the selected clock signals.

Timing Constraint Variables

- Four delay "variables" can be specified:
  - Input pin to output pin (through combinational logic)
  - Input pin to register (1/2 clock period typical)
  - Register to output pin (1/2 clock period typical)

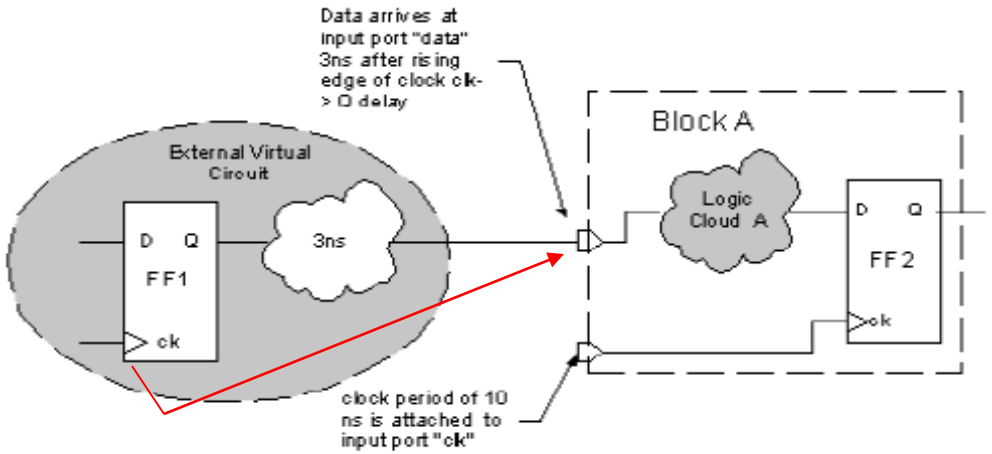
  Sum = 1 period: FF in one block to FF in another block
  - Register (from time it is clocked) to register (to meet setup time)

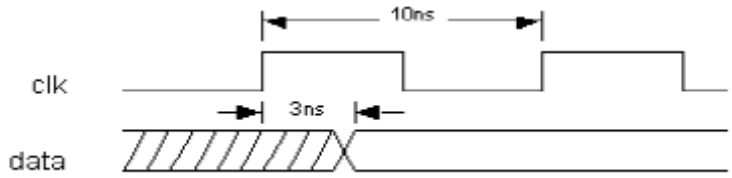
Examples: set register2register 8
set input2register 5



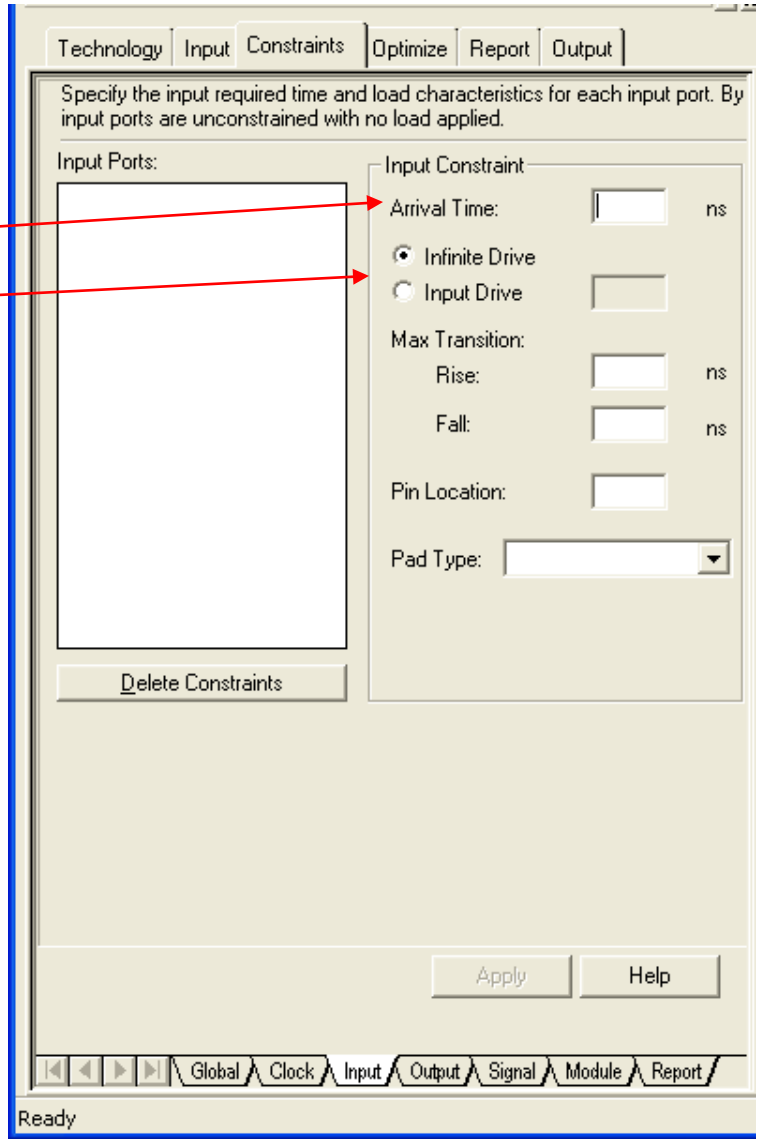
Input constraints

- Arrival time from previous ckt to input pin, relative to clock.
  - attribute: arrival_time
- Drive level of circuit driving the pin & max rise/fall time allowed
  – used in delay/loading calculations.



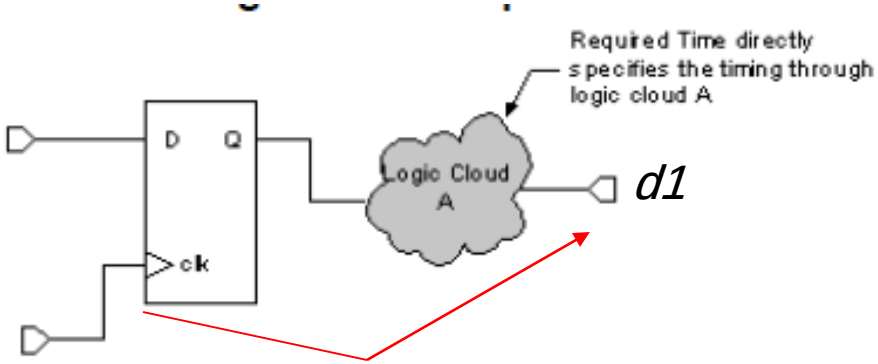


arrival_time 3 { data}

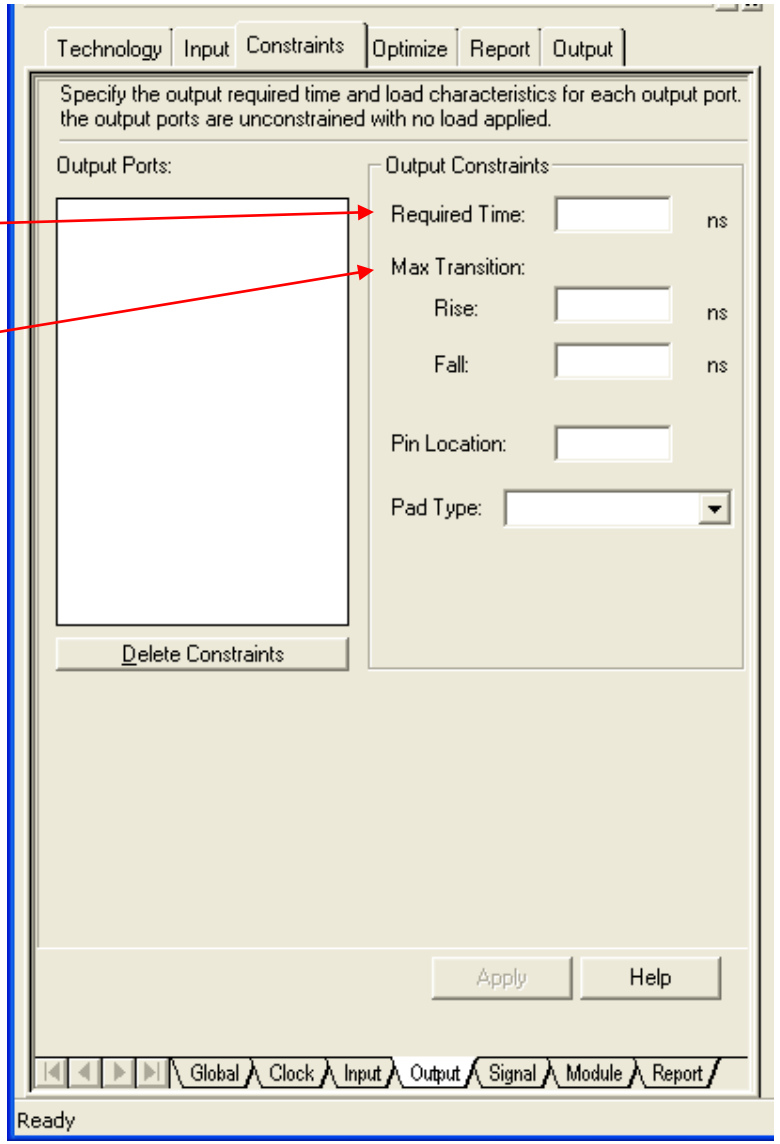


Output constraints

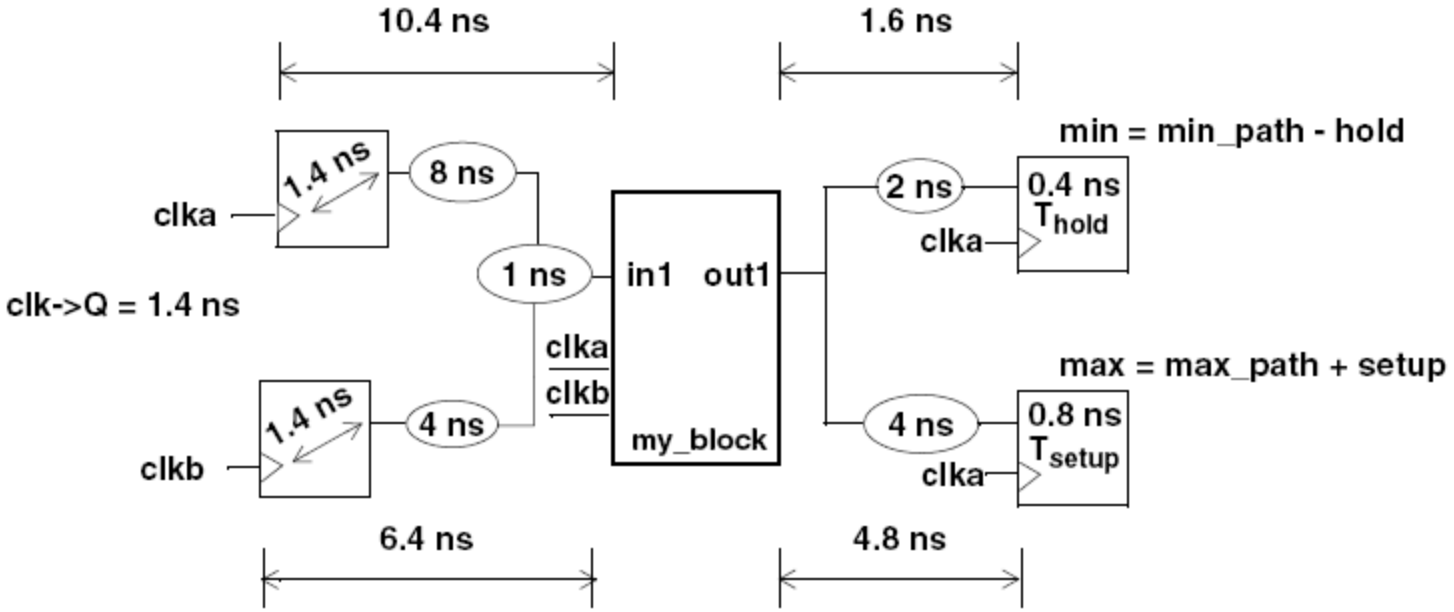
- Time from clock to valid output at pin, to be used by external ckt
  - attribute: required_time
- Max rise/fall time allowed
  – used in delay/loading calculations.



required_time 7 d1



Sequential circuit example



clock_cycle 20 clka
clock_cycle 30 clkb
arrival_time 10.4 -clock clka in1
arrival_time 6.4 -clock clkb in1
required_time 1.6 -clock clka -min out1
required_time 4.8 -clock clka -max out1

Required clock periods

Arrival at input pin from previous clock edge

Setup time of output pin from next clock edge

Module constraints

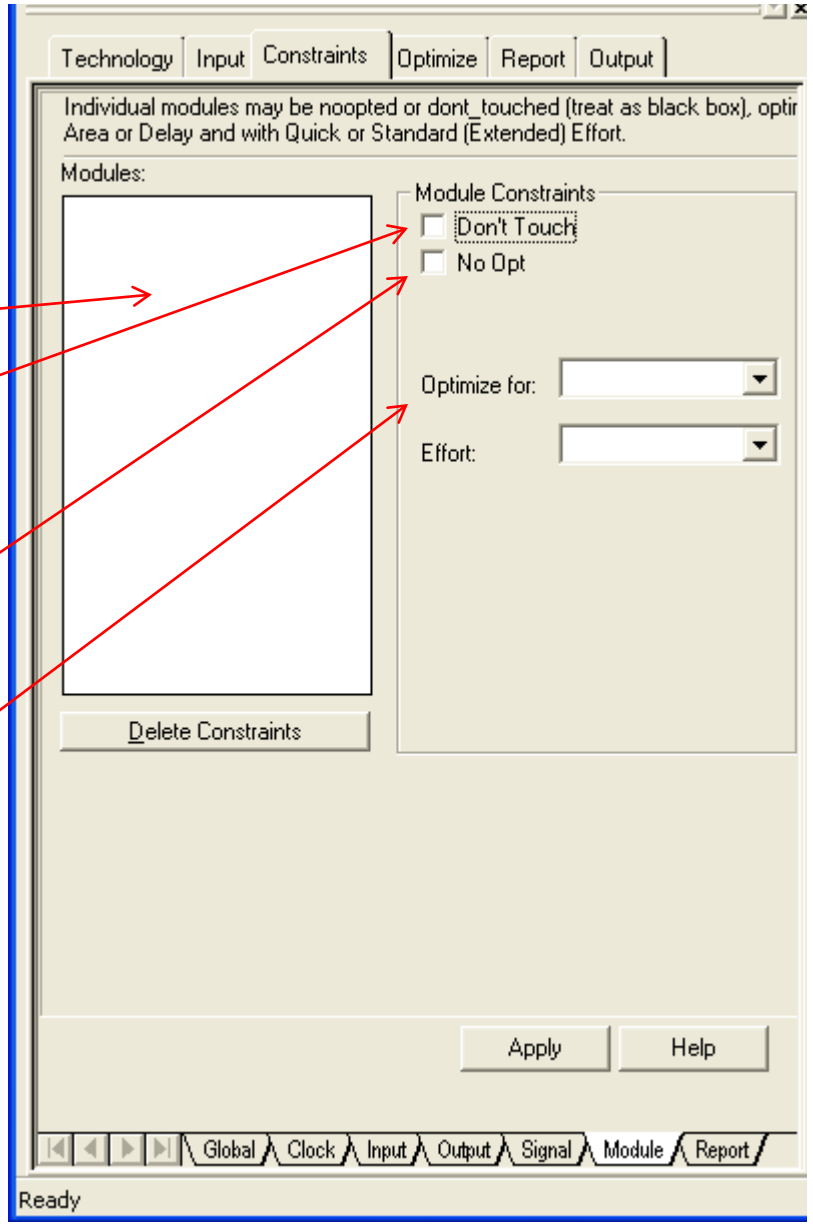
Commands/Attributes:
don't_touch, noopt

Select modules

Don't touch: keep module as-is, including lower levels

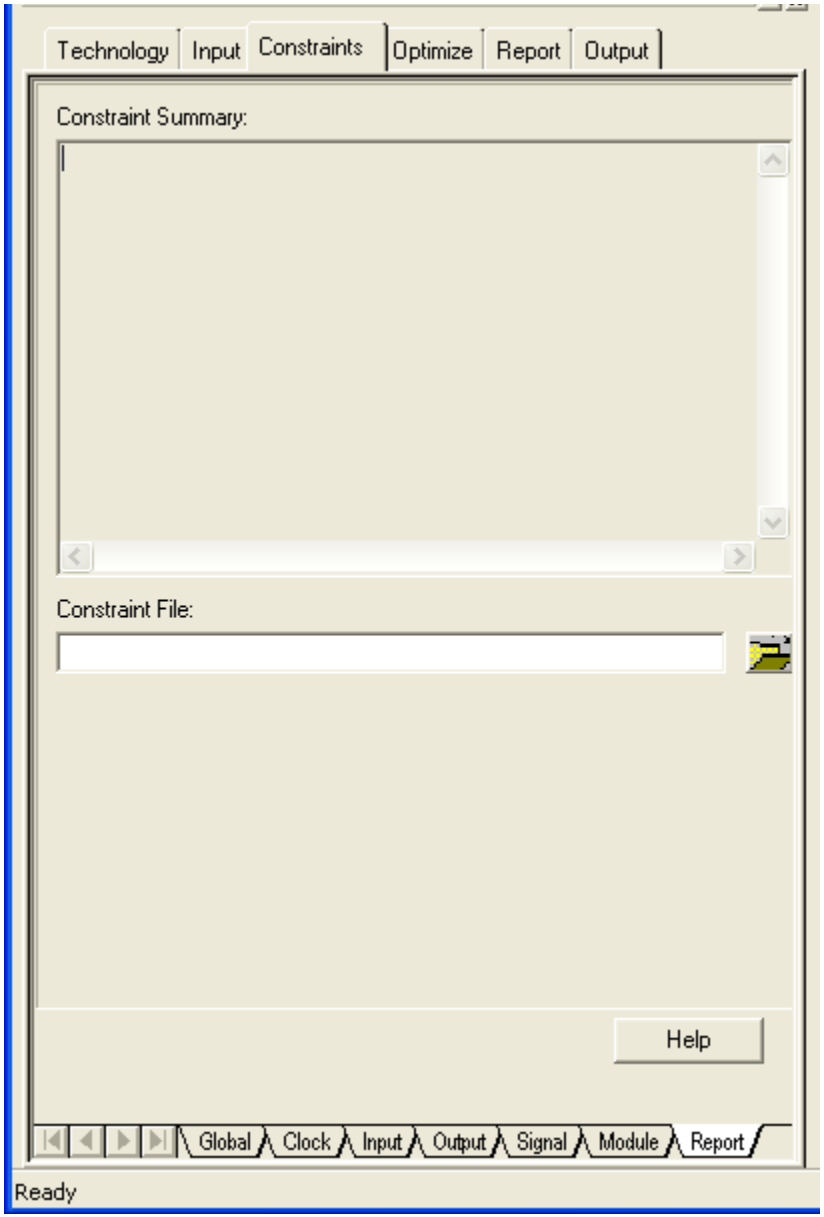
No Opt: keep selected module, but lower levels may be optimized

Select area/delay optimization and effort level for each module



Write a Constraint File

For subsequent synthesis and/or for physical place & route tool.



Pre-optimization step

- Technology-independent logic optimization
  - Always done as part of optimize command
- pre_optimize [<design_name>]

  [-common_logic] share operators/primitives with common inputs – used in different clock cycles

  [-unused_logic] remove logic that doesn't affect outputs

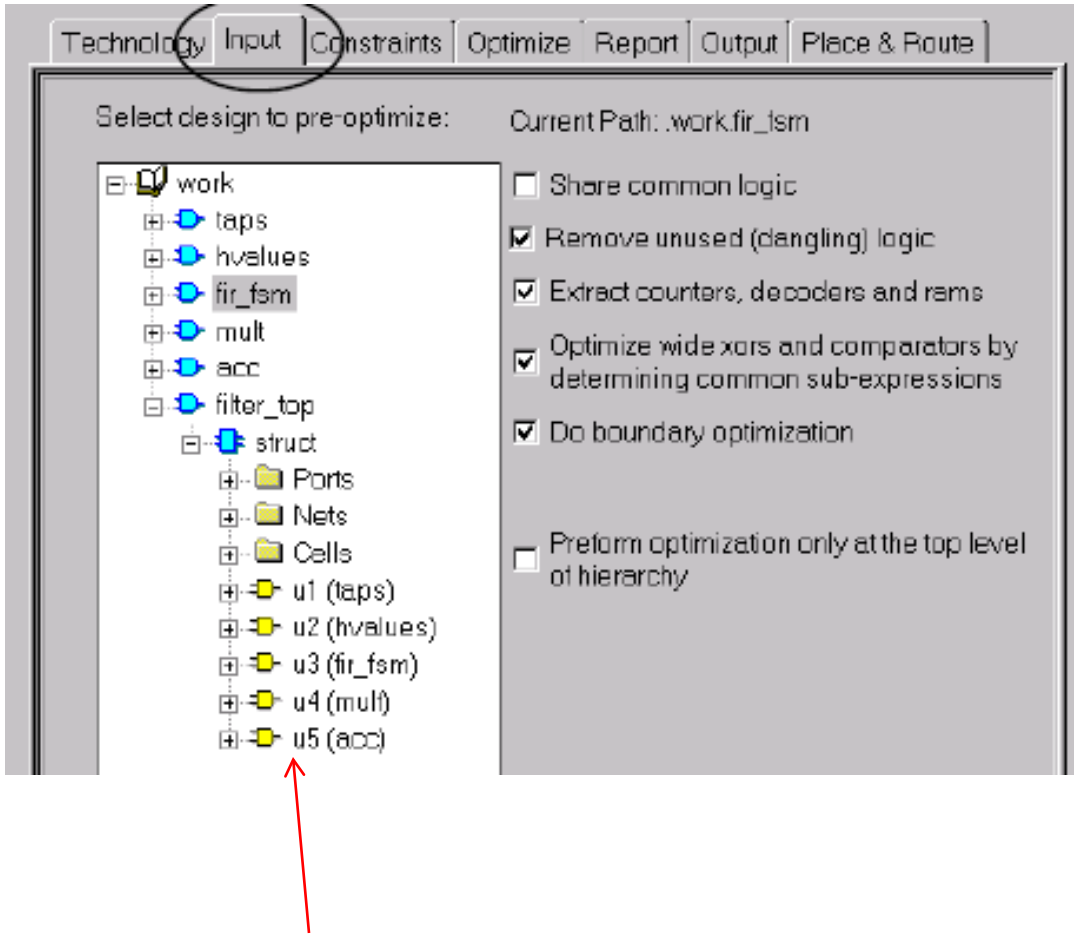
  [-extract] recognize counters, decoders, RAMs, ROMs

  [-xor_comparator_optimize] factor common sub-expressions from wide XORs/comparators

  [-single_level] do top level only (default is all levels)

  [-boundary] propagate constants (inputs tied high/low) from boundary, unused inputs, etc.

Pre-Optimize Power Tab



Select specific design to pre-optimize

Optimization step

- Control optimization via:
  - Timing constraints on I/O signals
  - Input drive and capacitance and output load
  - Definition on clocking schemes
- optimize <design> (default = current design)
  - -effort quick (one pass) | standard (multiple passes) | remap
  - -area | -delay | -auto [optimize area (default), delay or both]
  - -hierarchy preserve | flatten | auto
  - -macro | -chip [-macro = ASIC block; -chip includes I/O pads]
  - -single_level [optimize top only – otherwise all levels]
  - -target <tech> [default is to use loaded synth library]
  - -io_target <tech>

Timing Optimization

- **optimize_timing** [<design_name>]

  [**-through** <node_lists>] – opt through these nodes

  [**-force**] – use longest paths (ignores constraints)

  [**-single_level**] - top level only (o/w all levels)

  If no options: try to improve most critical paths within user-specified timing constraints

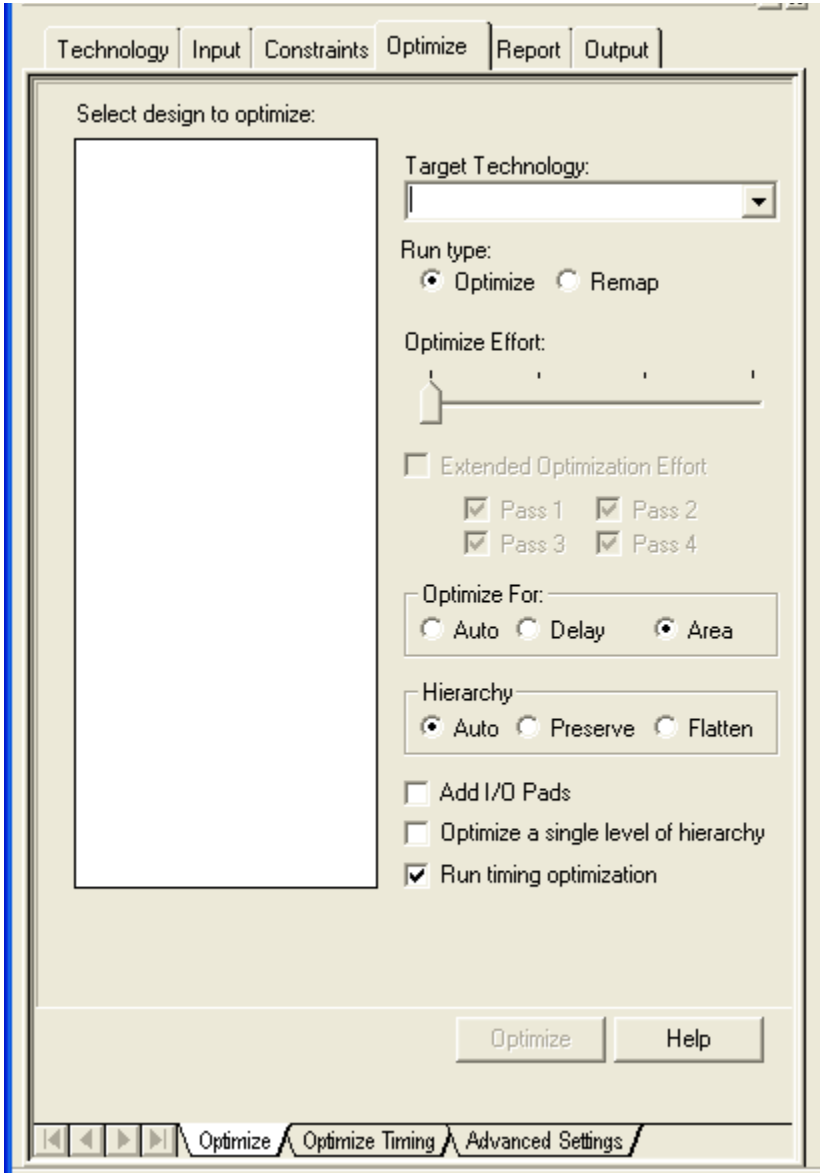
- Improve arrival_time at end of each critical path
- Most effective if user specifies constraints:
  - Input arrival, output required times, clock

Example: optimize all paths ending at signal *out1*

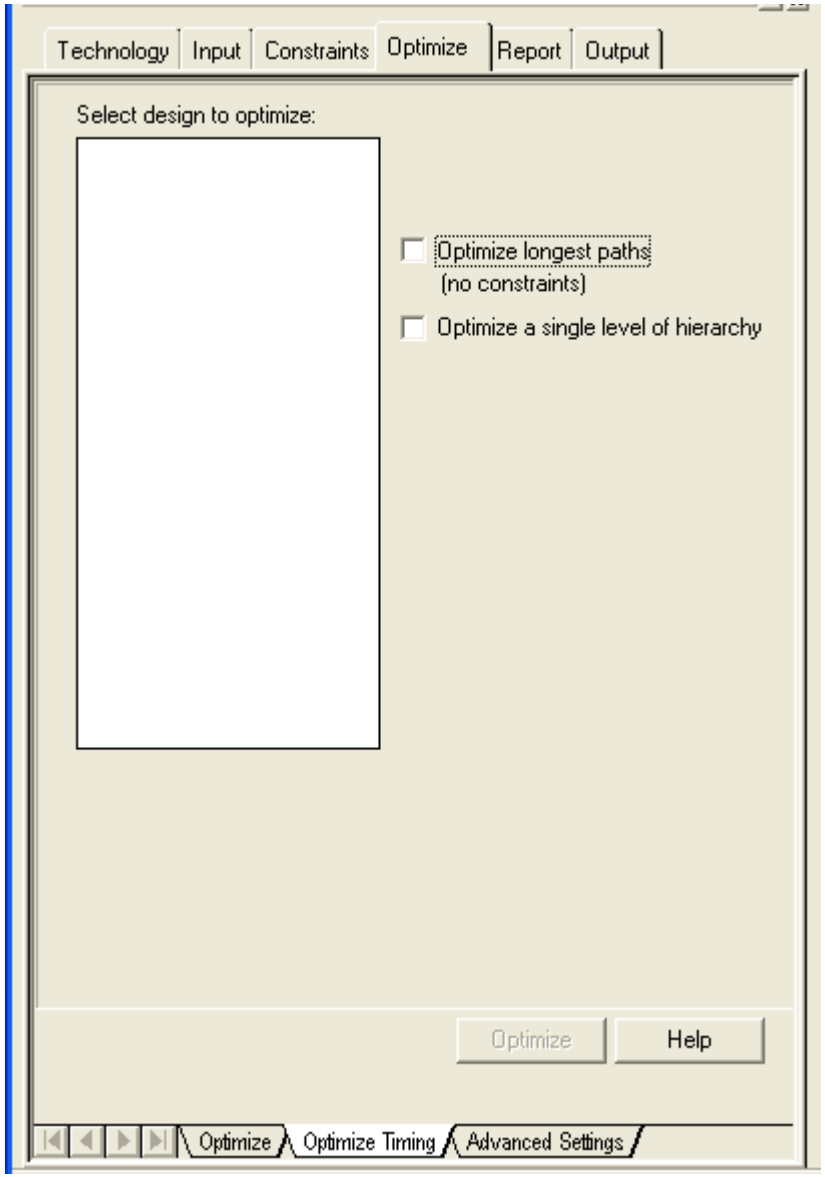
optimize_timing –through out1

Optimize Tab

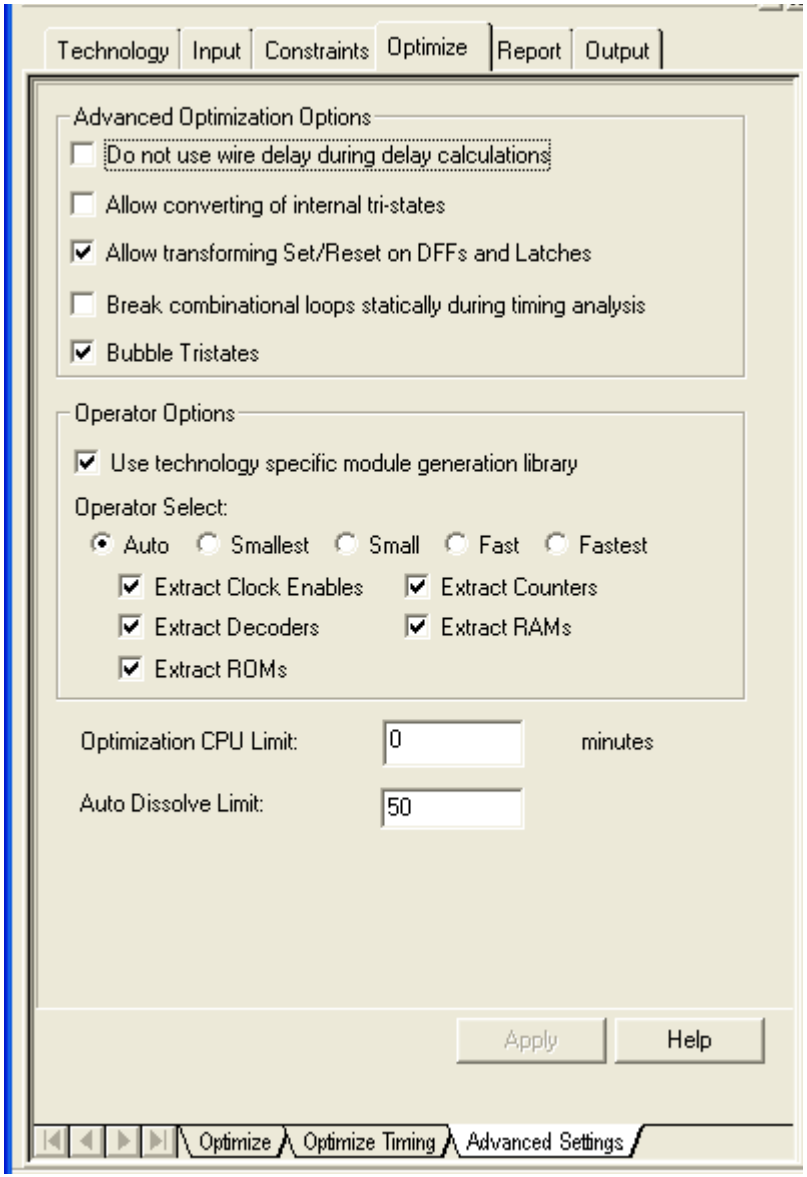
Set variables for both optimize and optimize_timing commands



“Optimize Timing”



“Advanced Settings”



Save design to file(s)

- **write** <file_name>

  [**-format** <format_name>]

  VERILOG (.v), VHDL (.vhd), SDF (.sdf), EDIF (.edf)

  [**-downto** <library_name>]

  - don't write details of cells/primitives in library

  [**-silent**] - no warnings or information messages

  [**-single_level**] -top level only (default is all levels)

  [**-design** <design_name>] -default is current design

Examples:

write –format VHDL mydesign.vhd

write –format Verilog mydesign.v

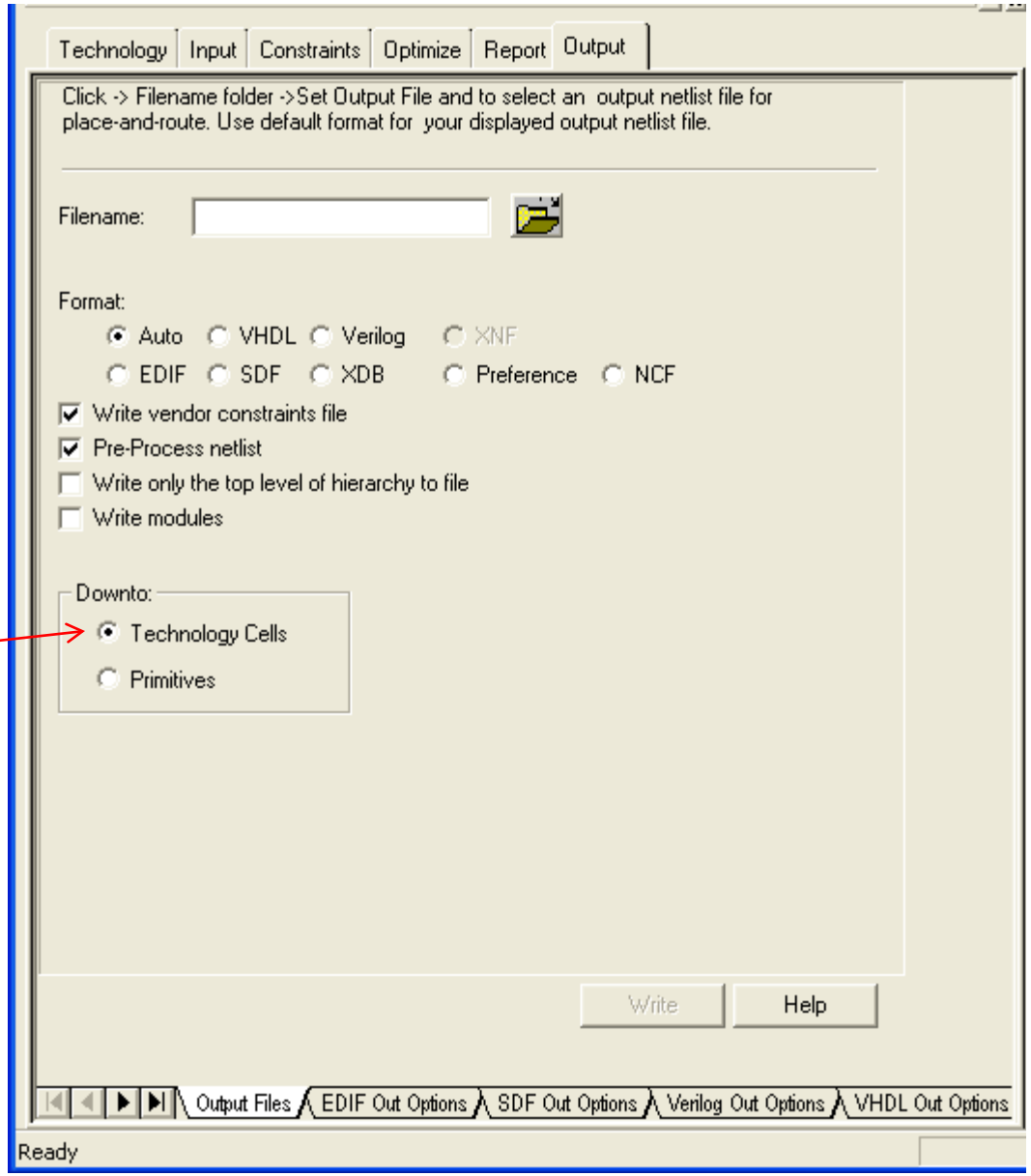
write –format SDF mydesign.sdf

(Delay parameters for timing analysis)

Output Tab

Write one or more files:
VHDL netlist
Verilog netlist
SDF (timing)

Select “Technology Cells” for circuit structure of cells from the technology library.



Area reports

- **report_area** [<report_file_name>]
  - [**-cell_usage**] – cell usage per instance in the design
  - [**-hierarchy**] – report each level of hierarchy separately
  - [**-all_leafs**] – report on all leaf cells, including black boxes

Example:

report_area -cell_usage –hierarchy mult_area.rpt

Generate a report showing all cell instances in each separate block in the design hierarchy. The report includes total number of primitive elements (std. cells) and equivalent area of each.

Delay reports – design timing information

- **report_delay** [<report_file_name>]
  - [**-num_paths** <number>] - default 10
  - [**-longest_path**] – sorted by longest first
  - [**-critical_paths**] – paths sorted by slack time
  - [**-end_points**] – report slack/arrival/required times at end points
  - [**-start_points**] – report slack/arrival/required times at start points
  - [**-clock_frequency**]
  - [**-no_io_terminals**]
  - [**-no_internal_terminals**] - only primary outputs
  - [**-show_input_pins**] - gate inputs
  - [**-show_nets**] - net names + gate (pin)
  - [**-through** <node_list>] - paths through given nodes
  - [**-from** <start_points>] - paths starting at given points
  - [**-to** <end_points>] - paths ending at given points
  - [**-not_through** <node_list>] - no paths through the given nodes
  - [**-show_schematic** <number>]

Examples next page

Delay report examples

report_delay -critical_paths -num_paths 1 -clock_frequency
-show_nets delay.rpt

Report the most critical path (-num_paths 1) and the max clock frequency, and list all nets along the path (default is to show gate outputs only).

report_delay -longest_path -num_paths 1 -show_nets
-to [list PRODUCT* DONE] outdelay.rpt

Report longest path from primary inputs or flip-flops to primary outputs PRODUCT* and DONE. (Asterisk matches any character string, so PRODUCT* matches PRODUCT(0), PRODUCT(1), …)

report_delay -longest_path -num_paths 1 -show_nets
-from [list MCAND* START] indelay.rpt

Report longest path from primary inputs MCAND* and START to flip flop inputs or primary outputs.