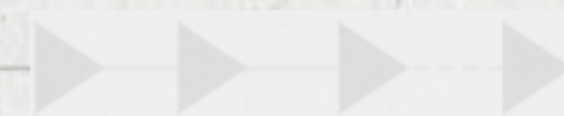


芯动力——硬件加速设计方法

第四章 逻辑综合(1)

邸志雄@西南交通大学

zxdi@home.swjtu.edu.cn



参考书目

郭炜等. SoC设计方法与实现 (第3版). 电子工业出版社.
出版时间: 2017-08-01.第八章.

Synopsys Design Compiler User Guide.

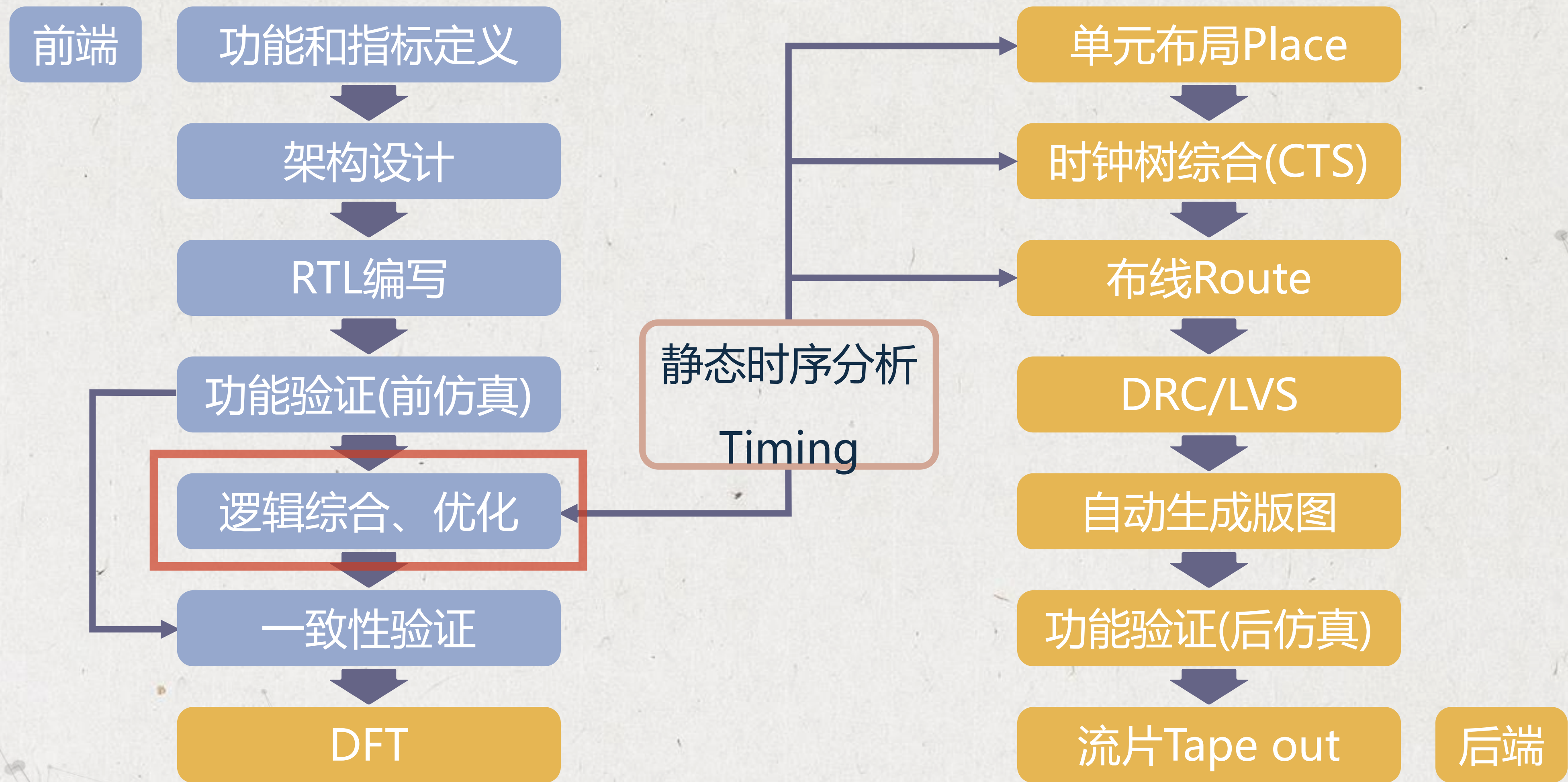
Rakesh ChadhaJ. Bhasker. Static Timing Analysis for
Nanometer Designs. Springer, 2009. Chapter-3.

学完本专题后，要掌握什么？

- ① 懂得为什么要逻辑综合;
- ② 懂得逻辑综合基本原理?
- ③ 懂得逻辑综合需要提供哪些文件?
- ④ 懂得逻辑综合过程中施加约束?
- ⑤ 懂得逻辑综合能够产生哪些结果。



回顾：芯片设计流程



概述

- 综合是前端模块设计中的重要步骤之一，综合的过程是将行为描述的电路、RTL 级的电路转换到门级的过程；
- Design Compiler 是 Synopsys 公司用于做电路综合的核心工具，它可以方便地将 HDL 语言描述的电路转换到基于工艺库的门级网表。
- 逻辑综合的目的：决定电路门级结构、寻求时序和与面积的平衡、寻求功耗与时序的平衡、增强电路的测试性。

本章将初步介绍综合的原理以及使用 Design Compiler 做电路综合的全过程。

逻辑综合的三个阶段

(逻辑综合)
Synthesis

=

(转译)
Translation

+

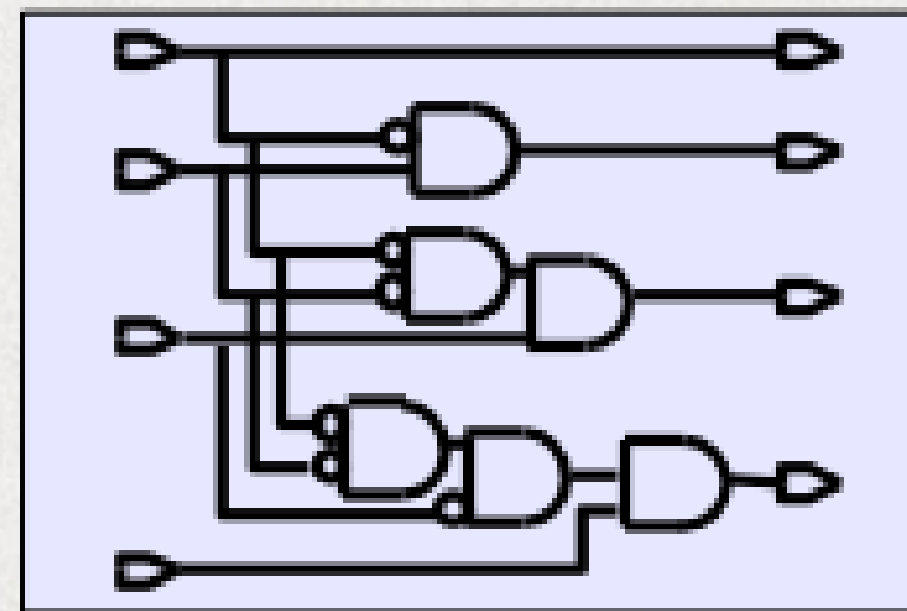
(优化)
Optimization

+

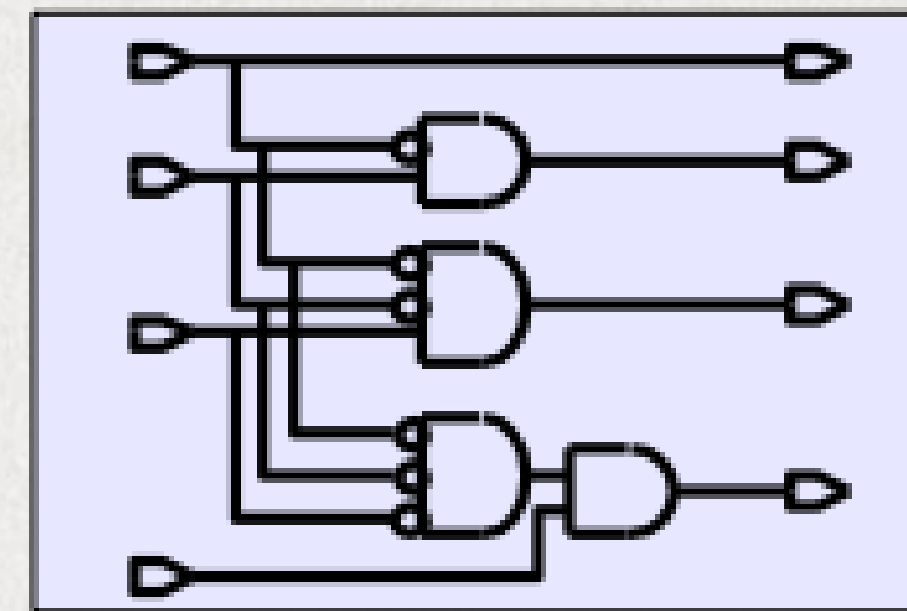
(映射)
Mapping

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

HDL Source



Generic Boolean
(GTECH)

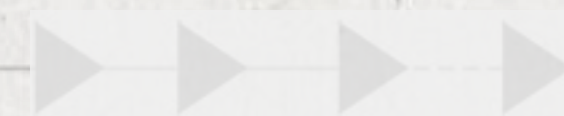


Target Technology

Translate

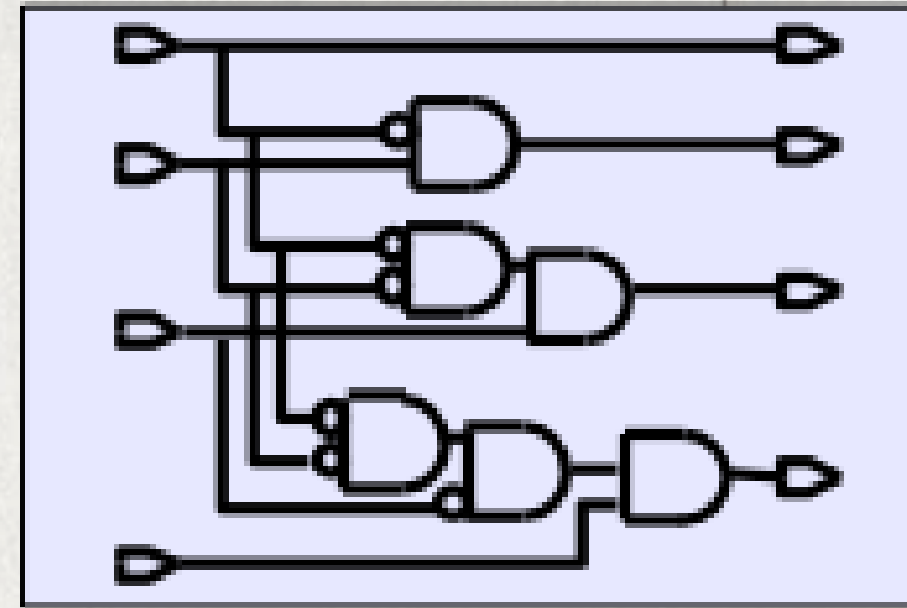
Optimize

Map

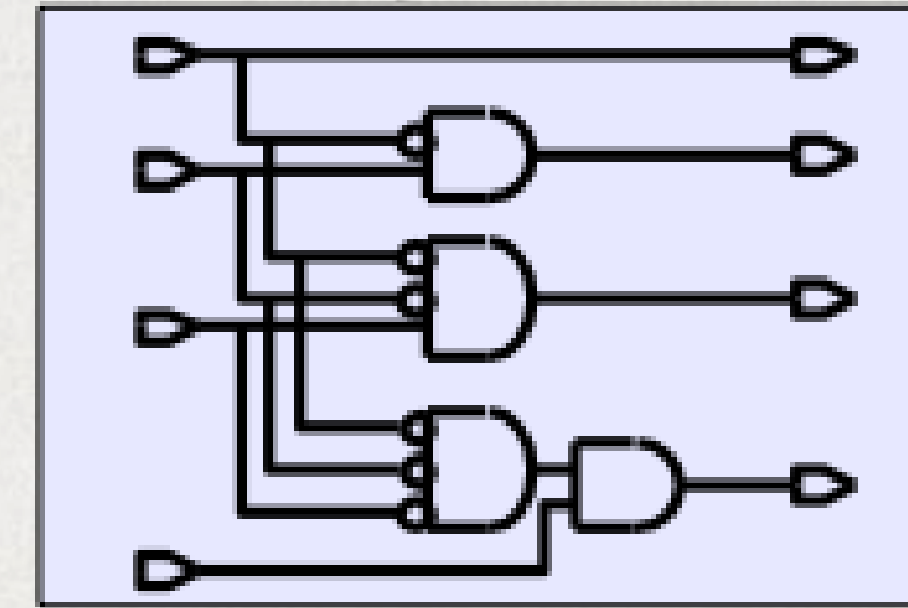



```
residue = 16' h0000;  
if (high_bits == 2' b10)  
  residue = state_table[index]; else  
  state_table[index] = 16' h0000;
```

HDL Source



Generic Boolean
(GTECH)



Target Technology

Translate

Optimize

Map

转译

- 这个数据库跟工艺库是独立无关的

优化

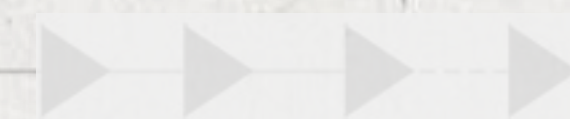
〔工作频率〕

〔面积〕

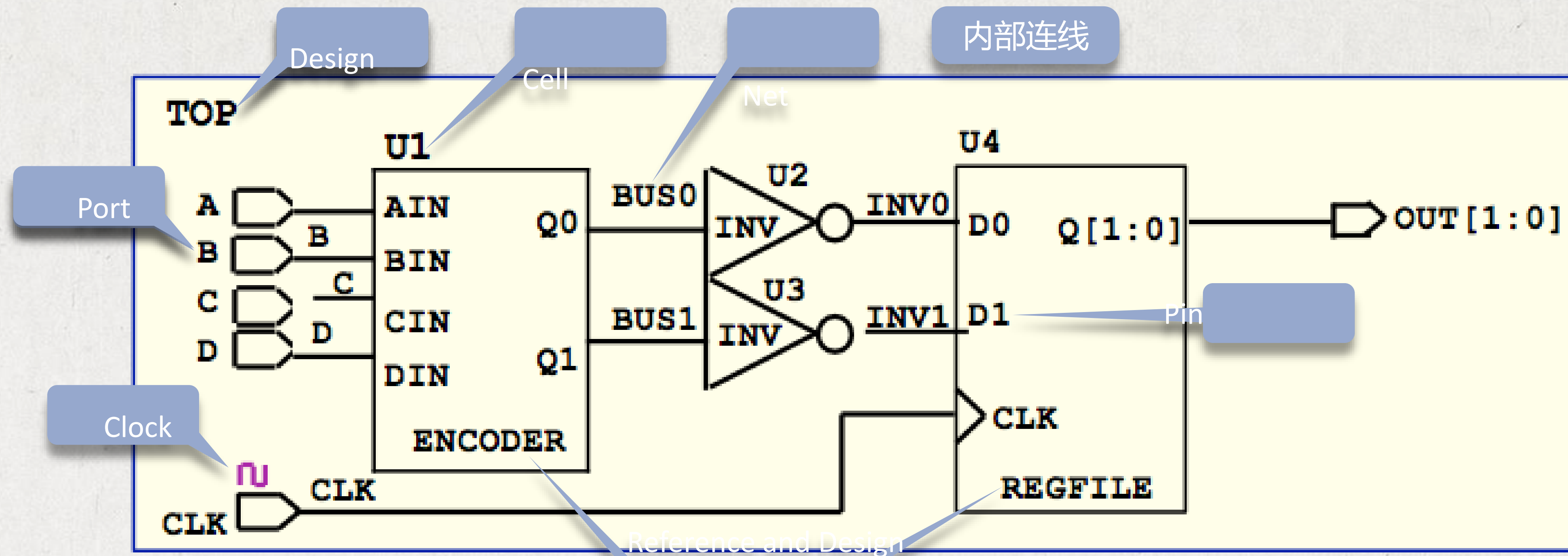
〔功耗〕

映射

- 最终形成该工艺库对应的门级网表



设计对象Design Objects



原电路

pin指的是电路中所有的引脚时钟的同步电路

只有最外部的端口称之为port

Designs: {ENCODER, REGFILE}

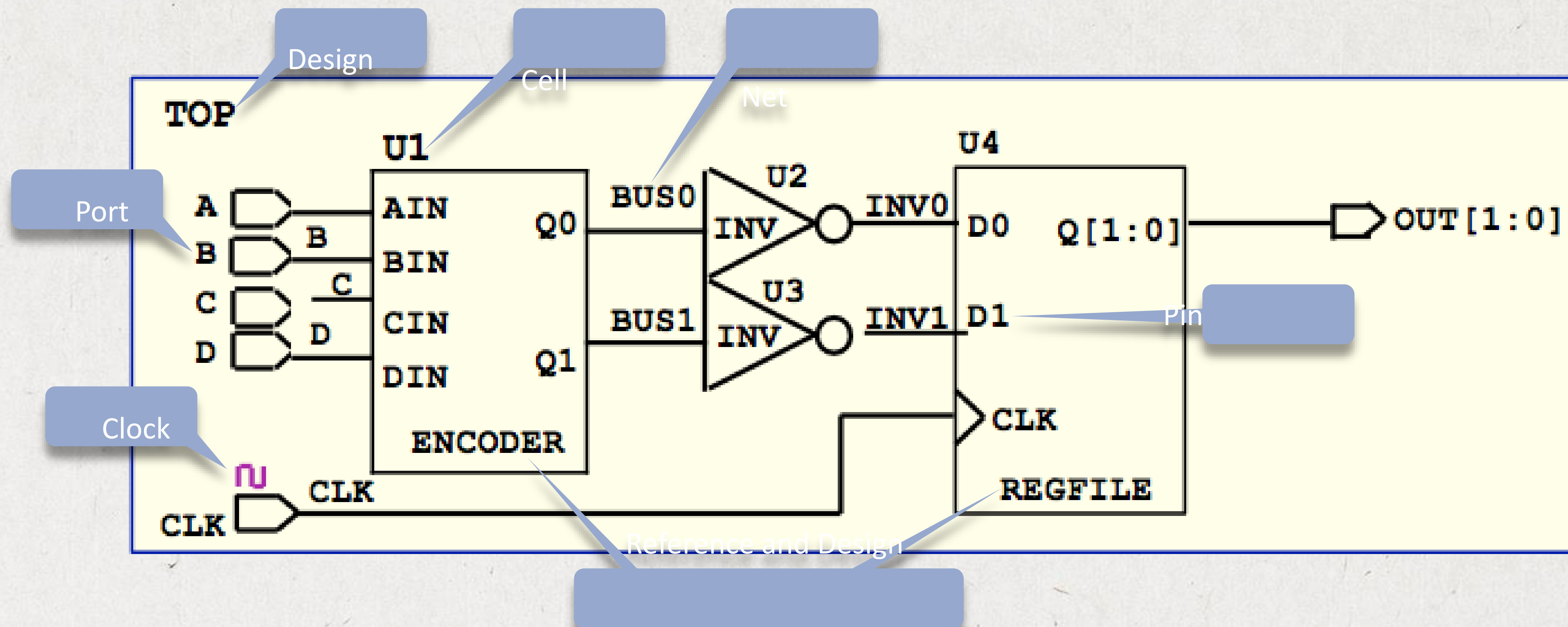
Ref是整体时钟问题会对电路造成重要的影响

被例化的模块

design指的是主要单独管理的对象

Cells: {U1, U2, U3, U4}

设计对象Design Objects




```

entity TOP is
    port (A, B, C, D, CLK: in STD_LOGIC;
        OUT1: out STD_LOGIC_VECTOR (1 downto 0));
end TOP;
architecture STRUCTURAL of TOP is
    ...
    signal INV1, INV0, BUS1, BUS0: STD_LOGIC;
begin
    U1: ENCODER port map (AIN=>A, . . . Q1=>BUS1);
    U2: INV port map (A => BUS0, Z => INV0);
    U3: INV port map (A => BUS1, Z=> INV1);
    U4: REGFILE port map (D0=>INV0, D1=>INV1, . . . CLK=>CLK);
end STRUCTURAL;

```

Design

Clock

Port

OUT1: out STD_LOGIC_VECTOR (1 downto 0));

end TOP;

architecture STRUCTURAL of TOP is

...

signal INV1, INV0, BUS1, BUS0: STD_LOGIC;

begin

Pin

Net

U1: ENCODER port map (AIN=>A, . . . Q1=>BUS1);

Cell

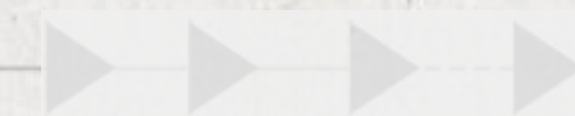
U2: INV port map (A => BUS0, Z => INV0);

U3: INV port map (A => BUS1, Z=> INV1);

U4: REGFILE port map (D0=>INV0, D1=>INV1, . . . CLK=>CLK);

end STRUCTURAL;

Reference



Design

```
module TOP (A,B,C,D,CLK,OUT1) ;
```

```
  input A, B, C, D, CLK;
```

Clock

```
  output [1:0] OUT1;
```

Port

```
  wire INV1,INV0,bus1,bus0;
```

Net

```
  ENCODER U1 (.AIN (A), . . . .Q1 (bus1)) ;
```

Reference

```
  INV U2 (.A (BUS0), .Z ( INV0)) ,
```

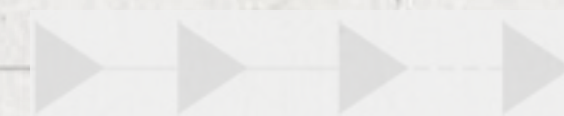
```
  U3 (.A( BUS1), .Z ( INV1)) ;
```

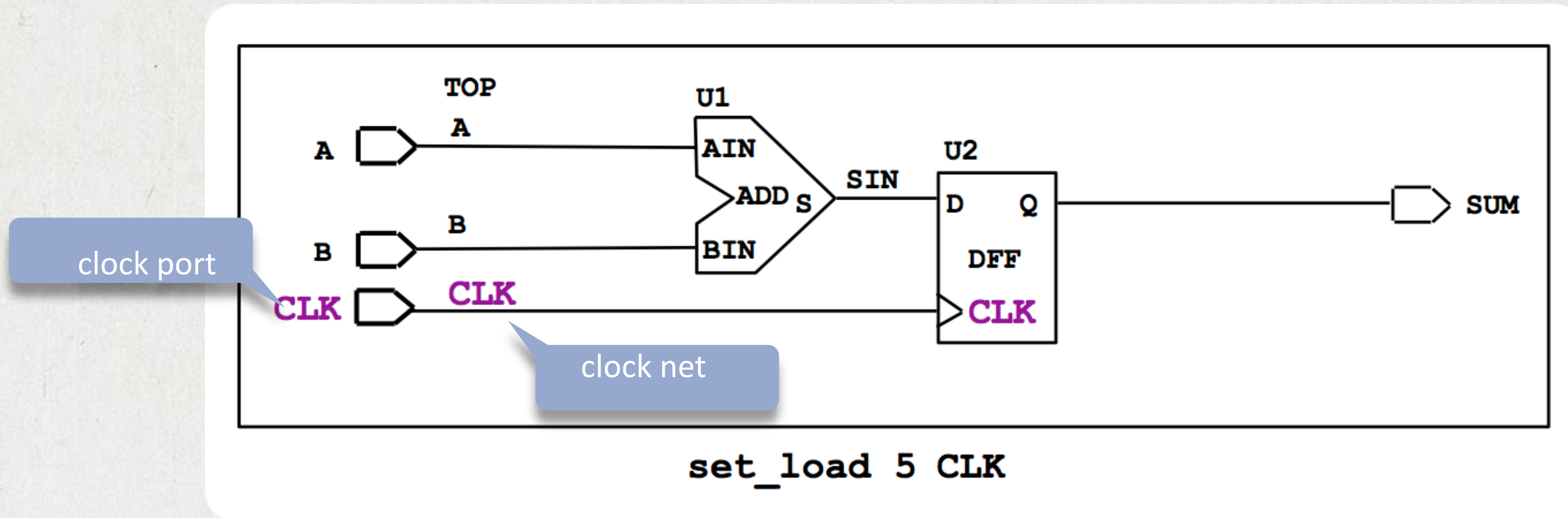
Cell

Pin

```
  REGFILE U4 (.D0 (INV0), .D1 (INV1), .CLK (CLK) ) ;
```

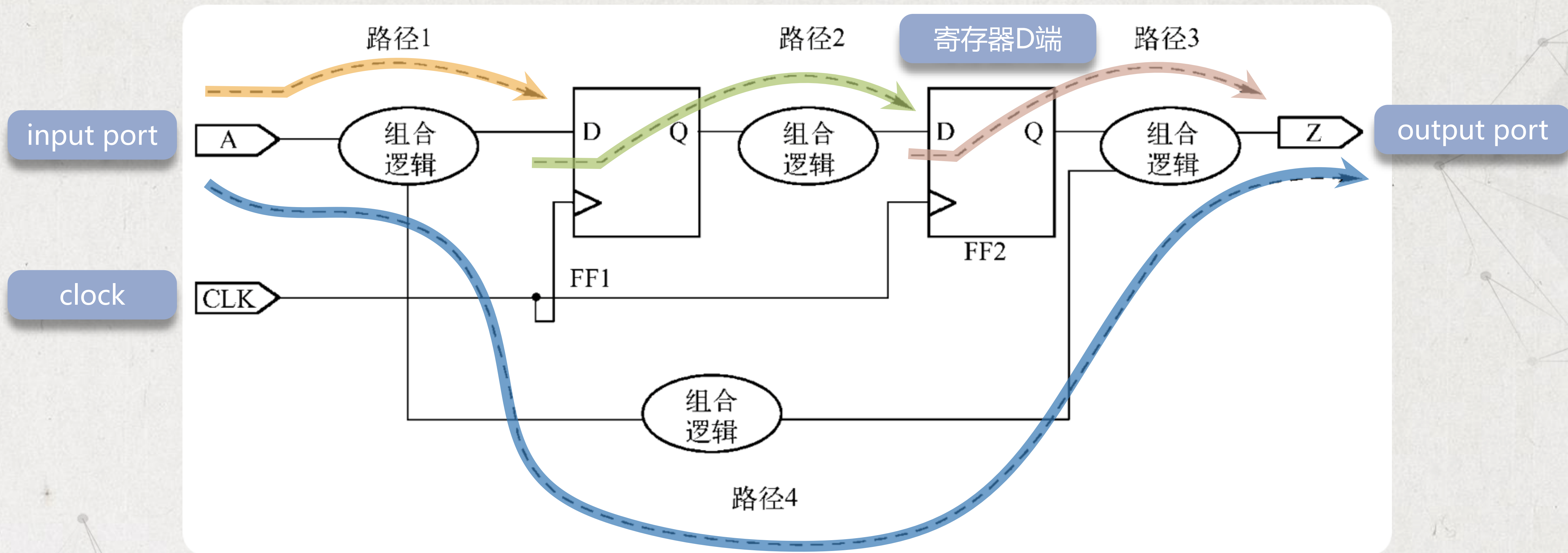
```
endmodule
```



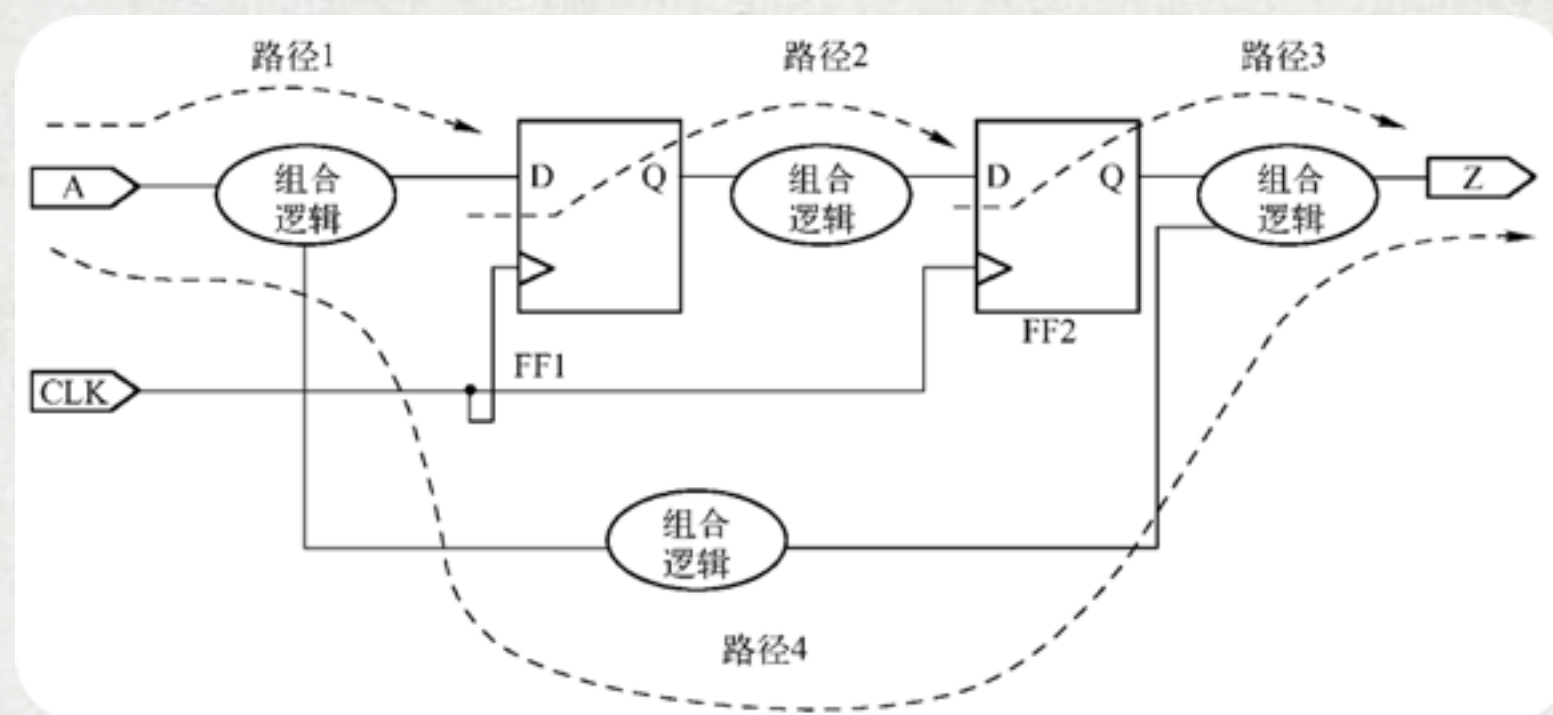


CLK” 指的是clock, port, net, or pin object?

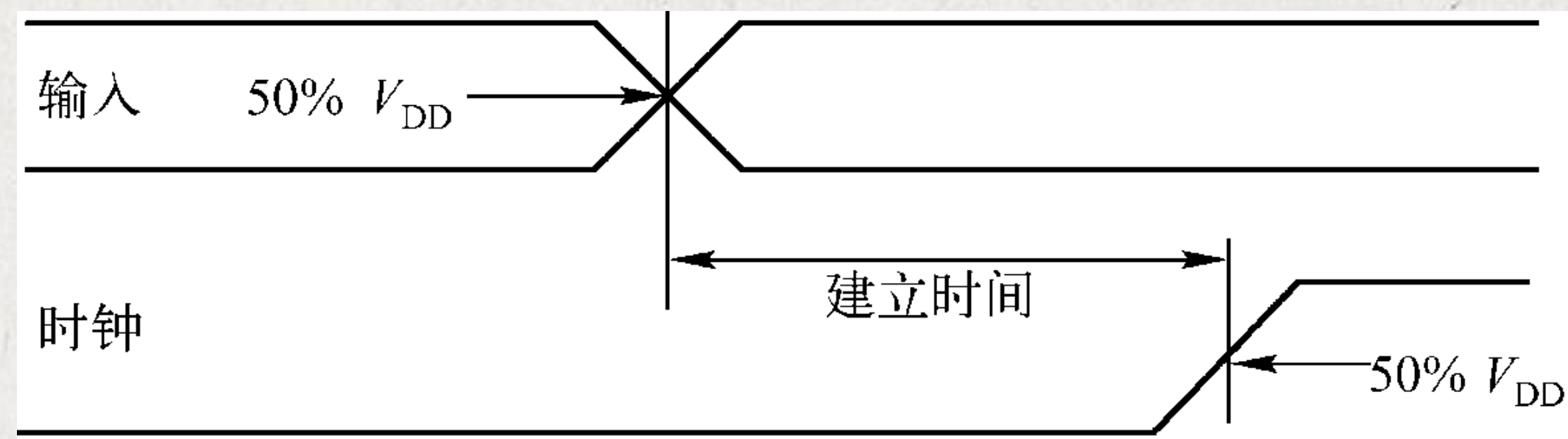
时序路径 (Timing Path)



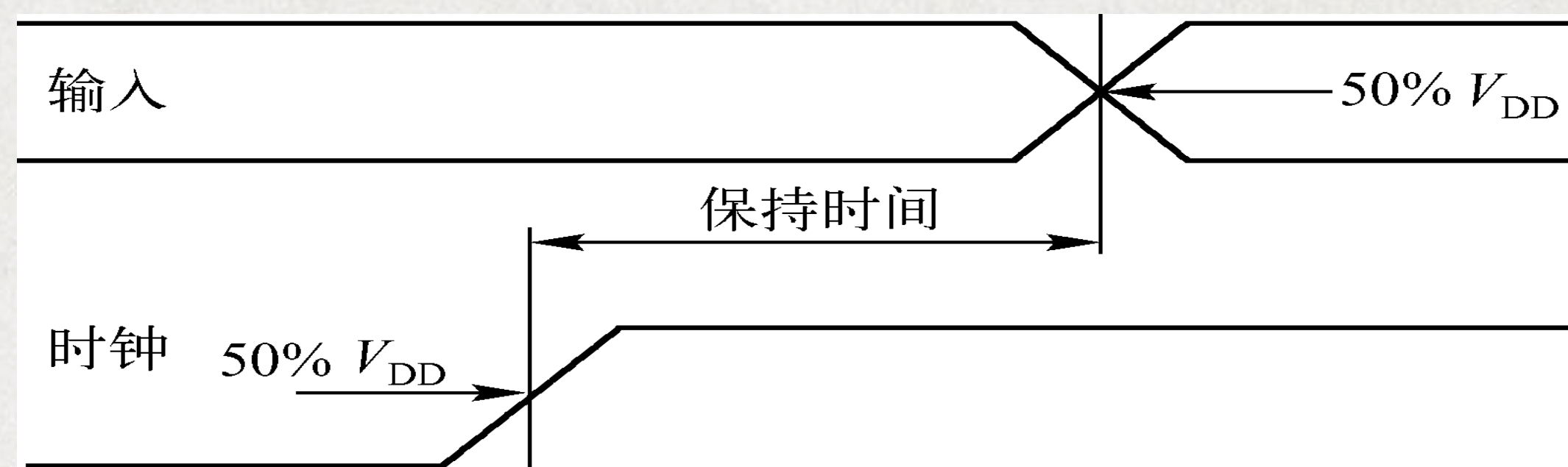
时序路径 (Timing Path)



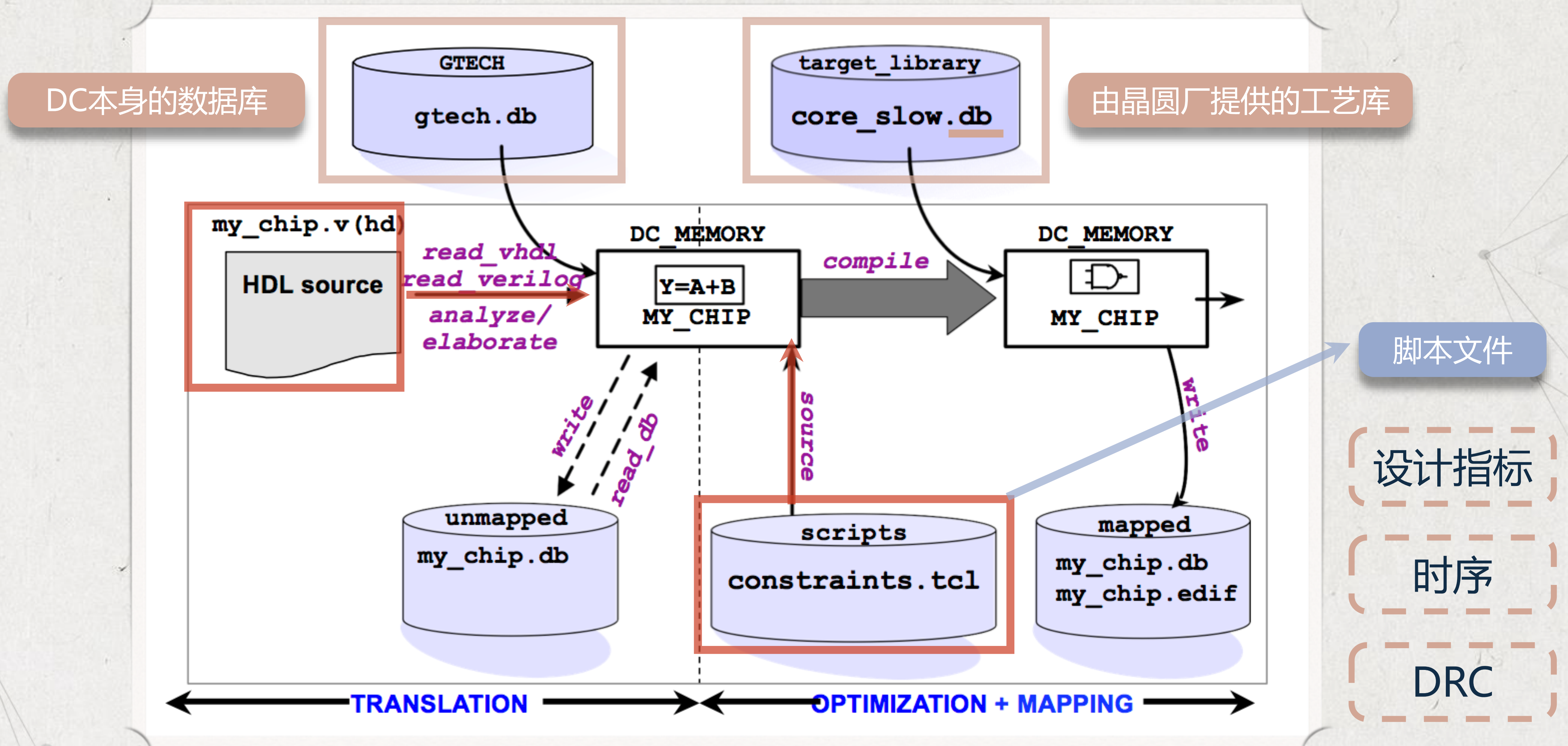
时钟上升沿之前数据需要稳定的时间



数据上升沿之后数据依然要保持稳定的时间



使用 Design Compiler 做综合的流程示意图



逻辑综合的实施流程

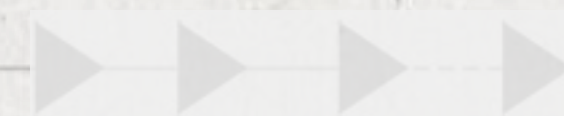
- 将具体介绍使用 Synopsys 公司的 Design Compiler 作综合的过程，整个过程大致可以分为下面几个部分：

预综合过程(Pre-synthesis Processes)

施加设计约束(Contrainting the Design)

设计综合(Synthesizing the Design)

后综合过程(Post-synthesis Process)

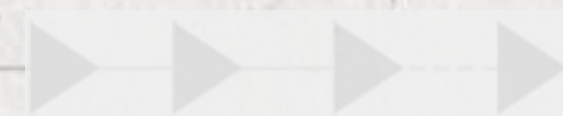




预综合过程(Pre-synthesis Processes)

在综合过程之前的一些为综合作准备的步骤

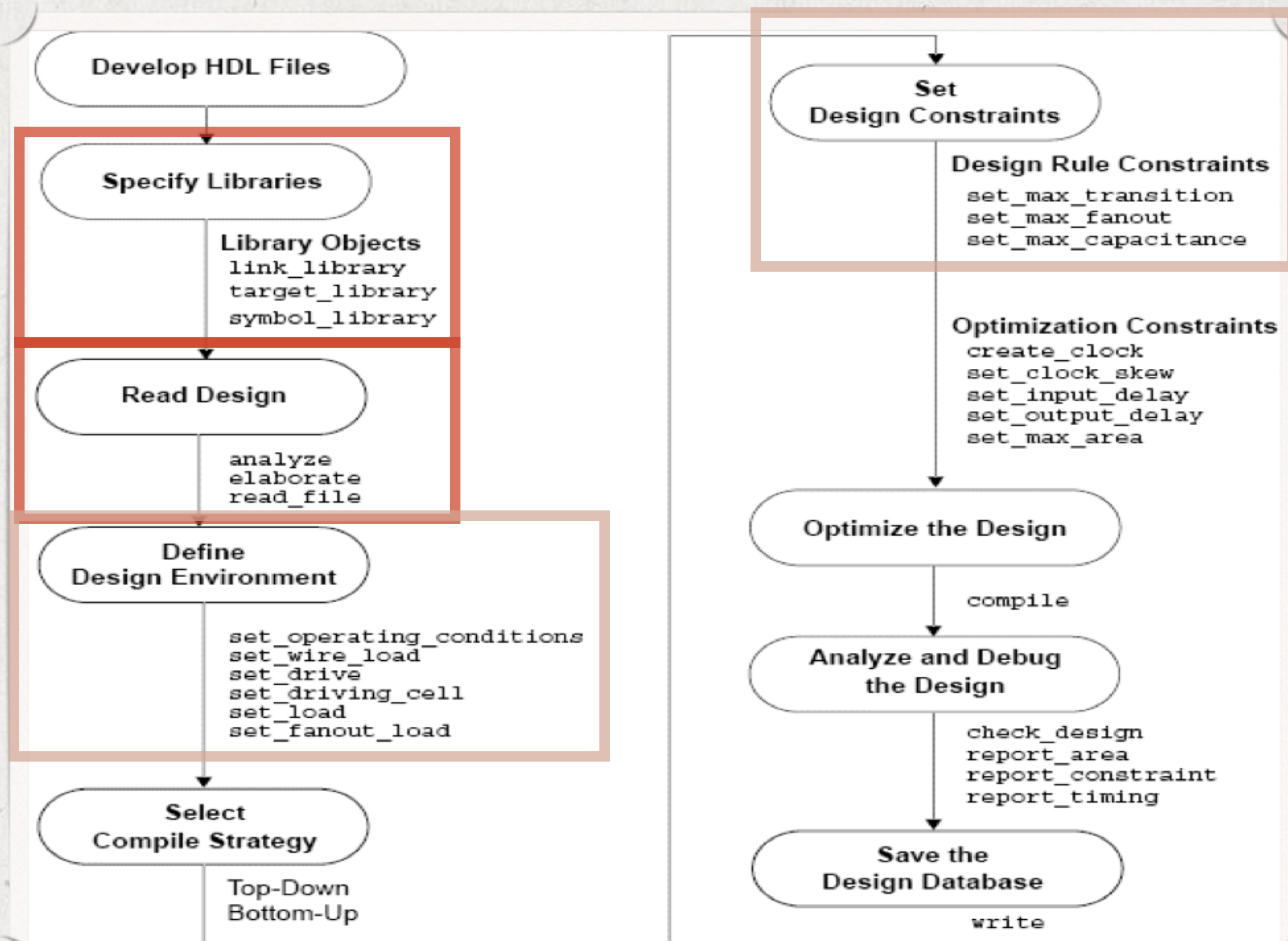
- Design Compiler 的启动
- 设置各种库文件
- 创建启动脚本文件
- 读入设计文件
- DC 中的设计对象
- 各种模块的划分
- Verilog 的编码



逻辑综合的实施流程

设计环境约束

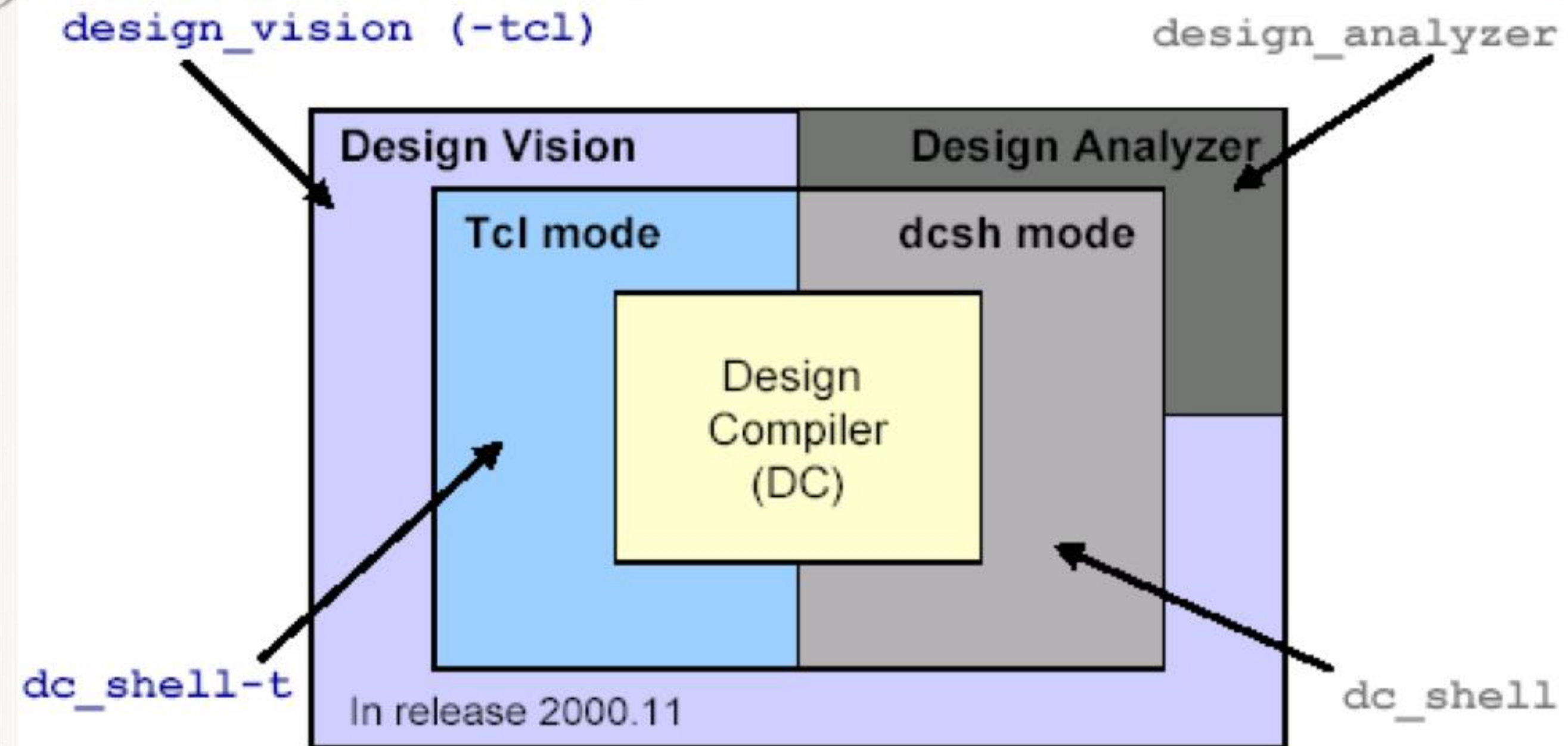
设计时序时序约束



Design Compiler 的启动

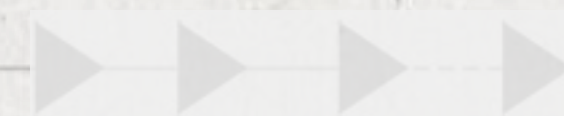
design_vision 图形方式

design_analyzer 图形方式



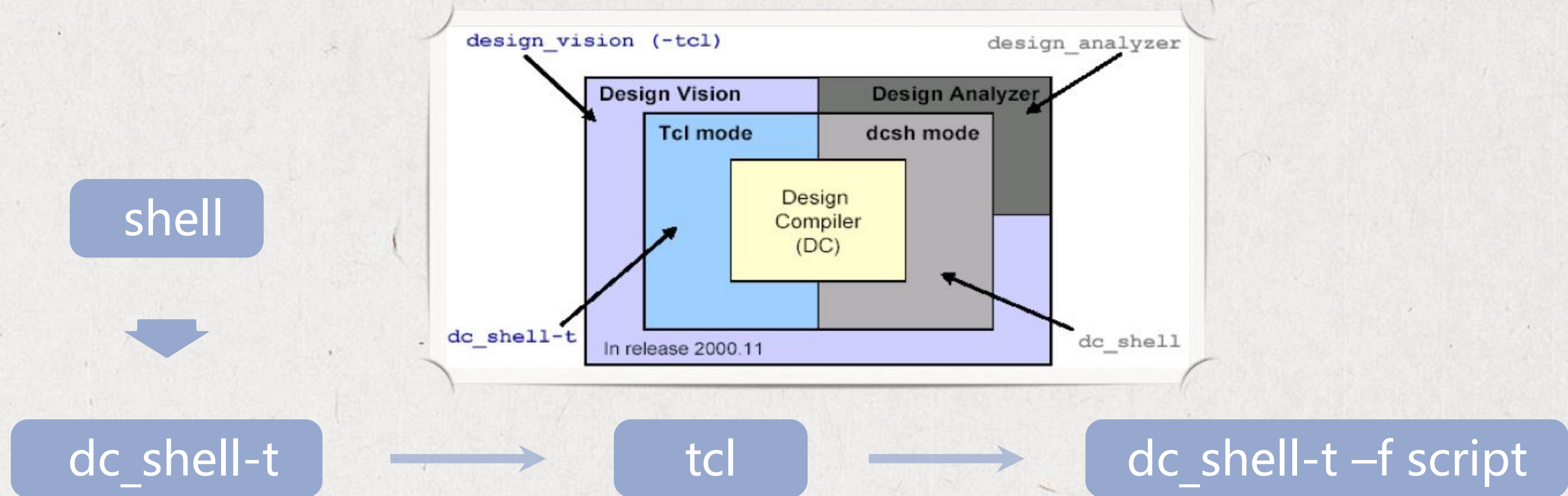
dc_shell-t 命令行方式

dc_shell 命令行方式



dc_shell-t 命令行方式

- 以 TCL (Tool Command Language) 为基础
- 在该脚本语言上扩展了实现 Design Compiler 的命令

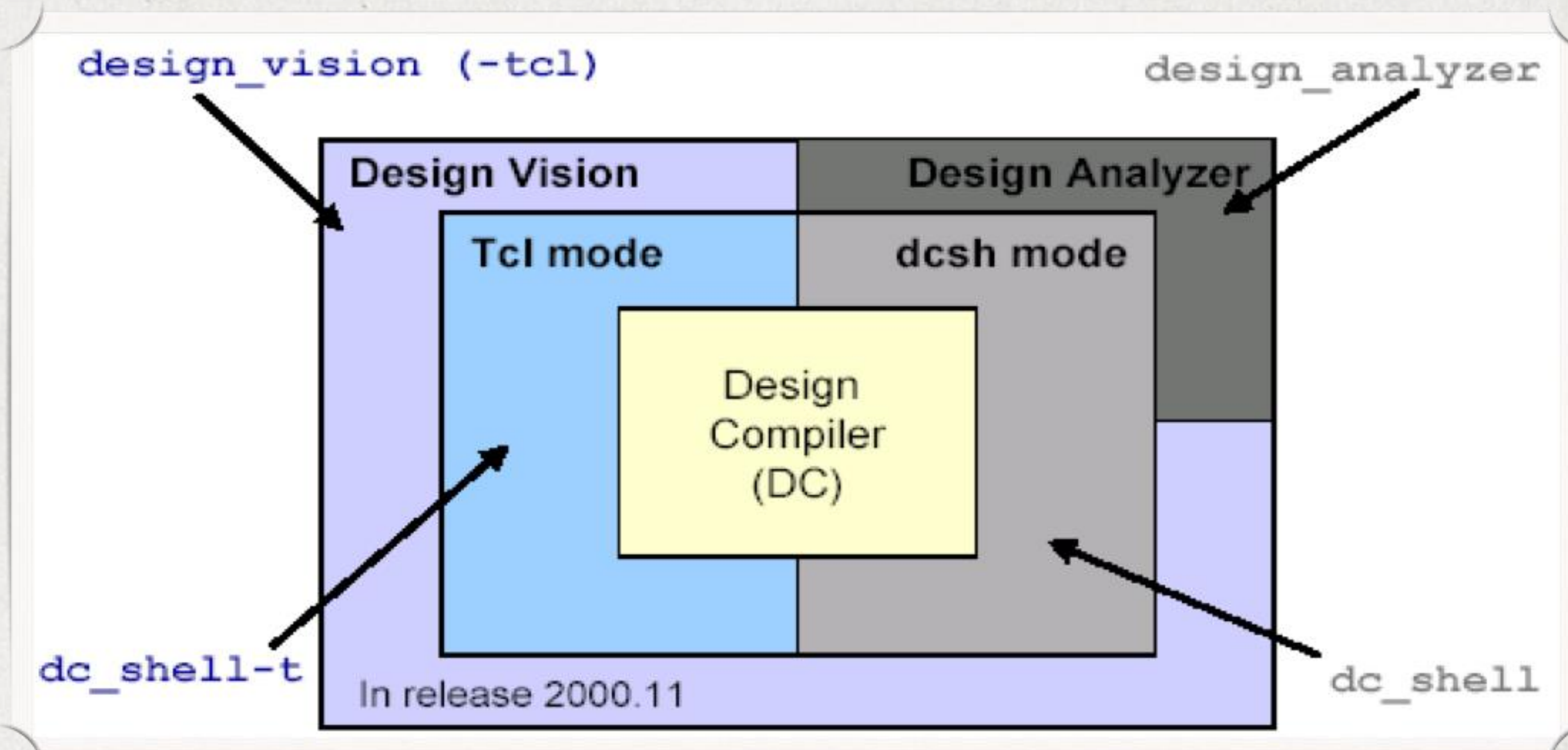


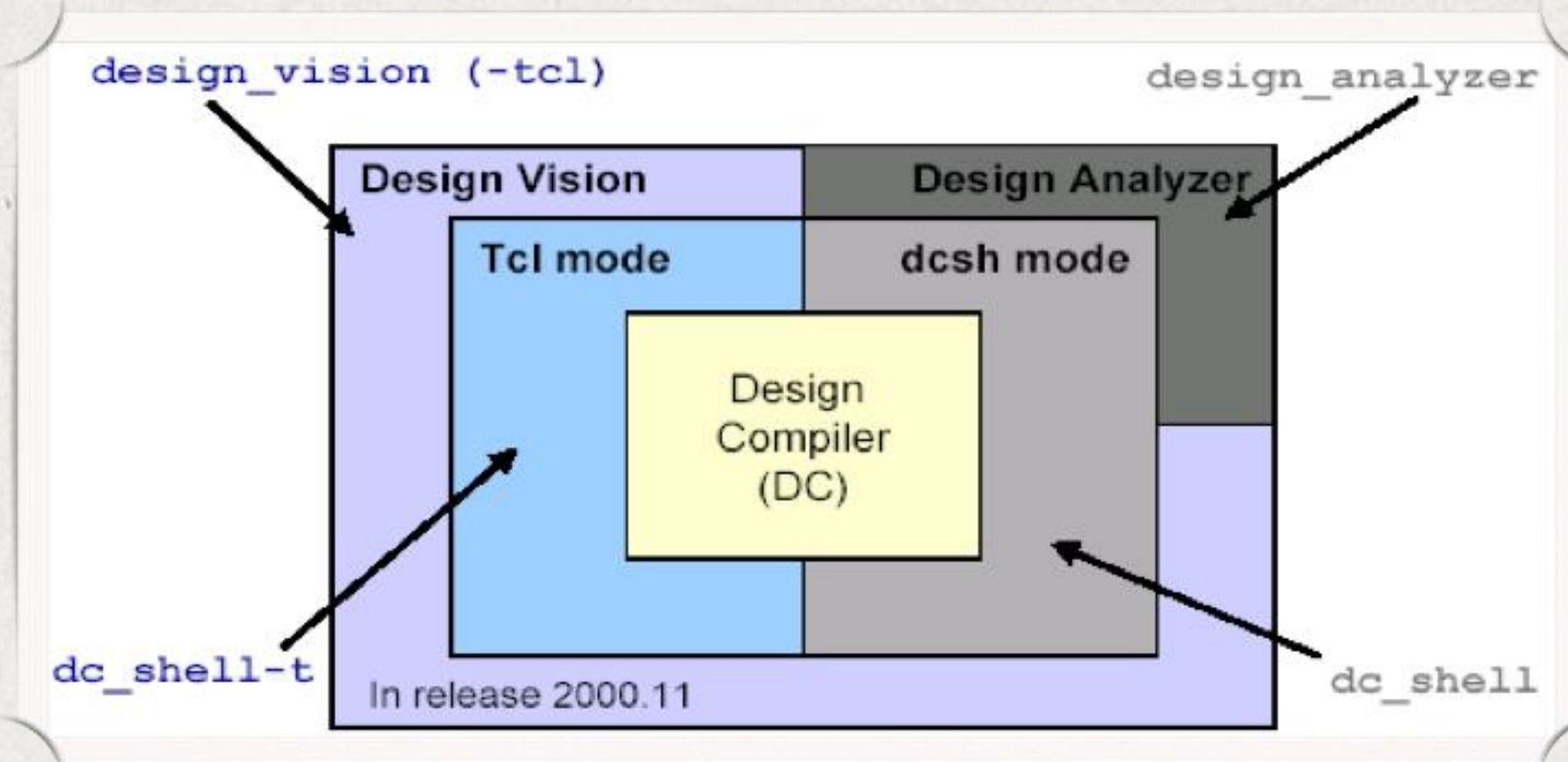
TCL 命令行方式

- 相对 shell 方式功能更强大
- 并且在 Synopsys 的其他工具中也得到普遍使用

design_vision 图形界面方式

- Design_vision 是与 tcl 对应的图形方式，用户可以在 shell 提示符下打“ design_vision” 来运行该方式。





command.log

view_command.log

- 用于记录用户在使用 Design Compiler 时所执行的命令以及设置的参数

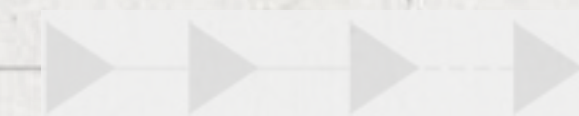
filenames.log

退出 design compiler 时会被自动删除

- 用于记录 design compiler 访问过的目录，包括库、源文件等

启动 dc_shell

只产生 command.log 的日志文件



读入设计文件

使用read指令来读入

注释：动作 1

```
read_verilog mychip.v
```

mychip.vhd

mychip.v

```
analyze -f vhd mychip.vhd
```

analyzed

mychip.syn
mychip.sim (VHDL)
mychip.mra

注释：动作 2

这一对指令要同时使用

```
elaborate MYCHIP
```

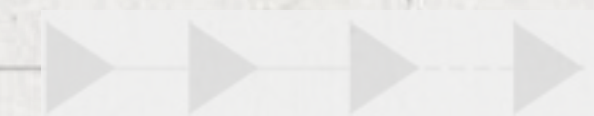
DC MEMORY

MYCHIP

```
write -output ./unmapped/mychip.db -hier
```

unmapped

mychip.db



Design Compiler 的read指令支持多种硬件描述的格式

.db

.v

.vhd

dcsh 工作模式

- 读取不同的文件格式只需要带上不同的参数

TCL 的工作模式

- 读取不同的文件格式需要使用不同的命令

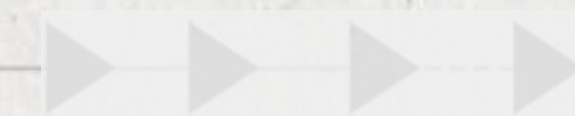
两种工作模式的读取命令的基本格式如下

read -format verilog[db、vhdI etc.] file //dcsh 的工作模式

read_db file.db //TCL 工作模式读取 DB 格式

read_verilog file.v //TCL 工作模式读取 verilog 格式

read_vhdl file.vhd //TCL 工作模式读取 VHDL 格式



两种工作模式的读取命令的基本格式如下

`read -format verilog[db、vhdl etc.] file` //dcsh 的工作模式

`read_db file.db` //TCL 工作模式读取 DB 格式

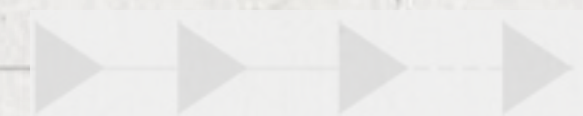
`read_verilog file.v` //TCL 工作模式读取 verilog 格式

`read_vhdl file.vhd` //TCL 工作模式读取 VHDL 格式

- Design Compiler 可以读取设计流程中任何一种数据格式

〔行为级的描述〕 〔RTL 级的描述〕 〔门级网表〕

- 不过由于不同的数据格式使得 Design Compiler 综合的起点不同，即使实现相同的功能，也可能会有不同的结果。



读取源程序的另外一种方式

analyze

- 分析 HDL 的源程序并将分析产生的中间文件存于 work (用户也可以自己指定)的目录下

elaborate

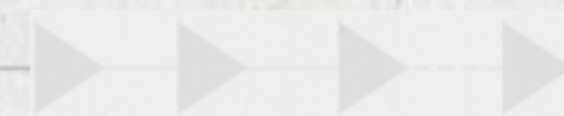
- 在产生的中间文件中生成 verilog 的模块或者 VHDL 的实体

缺省情况下

- elaborate 读取 的是 work 目录中的文件

analyze&elaborate

- 允许设计者在设计的GTECH建立之前，首先去分析设计的语法错误和进行HDL代码转换。

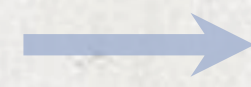


analyze&elaborate

- 允许设计者在设计的GTECH建立之前，首先去分析设计的语法错误和进行HDL代码转换。

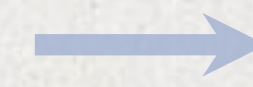
Analyze

做语法的检查



".syn" 文件

work路径下的
定义设计库内



elaborate

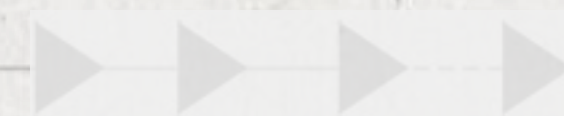
analyzed过的设计

- 只需用elaborate重新输入，节省时间。


read

- 不行

读入RTL代码



类别	analyze&elaborate	read
格式	verilog 或VHDL	verilog 、 VHDL、 EDIF、 db等所有格式
用途	综合verilog 或VHDL的RTL设计	读网表，设计预编译
设计库	用-library选项定义设计库名，存储“.syn” 文件	用缺省的设置，不能存储中间结果
Generics(vhdl)	可以对parameter进行操作	不能对parameter进行操作
Architecture(vhdl)	可以进行结构化的操作	不可用

- 
- 当读取完所要综合的模块之后，需要使用 link 命令将读到 Design Compiler 存储区中的模块或实体连接起来

link 命令

unresolved design reference 的警告信息

- 需要重新读取该模块
 - 在.synopsys_dc.setup 文件中添加 link_library，告诉 DC 到库中去找这些模块
- 同时还要注意 search_path 中的路径是否指向该模块或单元电路所在的目录。

