数字集成电路静态时序分析基础

邸志雄 博士, zxdi@home.swjtu.edu.cn

西南交通大学信息科学与技术学院

CONTENT

Part-1:TCL语言

TCL的应用:1-Overview

Tcl = **T**ool **C**ommand **L**anguage

A simple programming language.

Open, industry standard.

Many Synopsys tools' command interfaces are based on the Tcl command language.

TCL的应用:1-Overview

Tcl Scripts are Based on Commands

Builtin Commands → Provided by the Tcl interpreter itself

Application Commands → Provided by Synopsys tools, written as a command procedure in C or C++ using the Tcl extension mechanism

User Defined Commands → Provided by user, written as a command procedure in Tcl

TCL的应用:1-Overview

Example : Why Synopsys TCL?

How many clocks have been defined for this design?

Suggest a report that will list all clocks?

TCL的应用

Example : Why Synopsys TCL?

```
pt_shell> report_clock
Clock            Period   Waveform        Attrs     Sources
-------------------------------------------------------------
IO_PCI_CLK       15.00    {0 7.5}           p, G    {I_CLOCK_GEN/fb_clk}
IO_SDRAM_CLK      7.50    {0 3.75}          p, G    {I_CLOCK_GEN/fb_clk}
PCI_CLK          15.00    {0 7.5}           p, G    {I_CLOCK_GEN/clk}
SDRAM_CLK         7.50    {0 3.75}          p, G    {I_CLOCK_GEN/clk}
SD_DDR_CLK        7.50    {0 3              p, G    {sd_CK}
SD_DDR_CLKn       7.50    {3.75 7.5}        p, G    {sd_CKn}
SYS_2x_CLK        5.00    {0 2.5}           p, G    {I_CLOCK_GEN/clk_2x}
SYS_CLK          10.00    {0 5}                     {I_CLOCK_GEN/clk_1x}

. . .
```

31

32

33

Ridiculous! Writing a Perl script would be simpler!

TCL的应用:1-Overview

Example : Why Synopsys TCL?

sizeof_collection [all_clocks]

€ 52

Powerful

Flexible

Easy

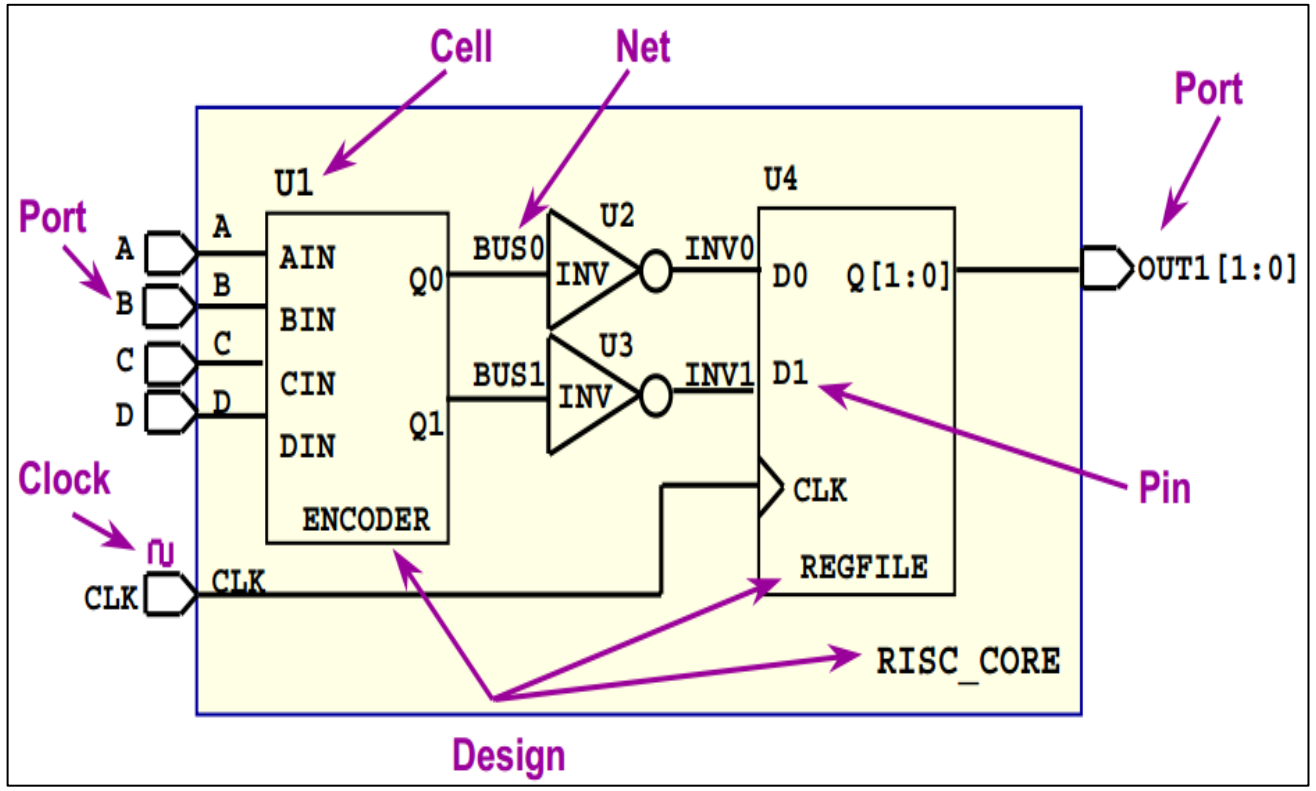
References:

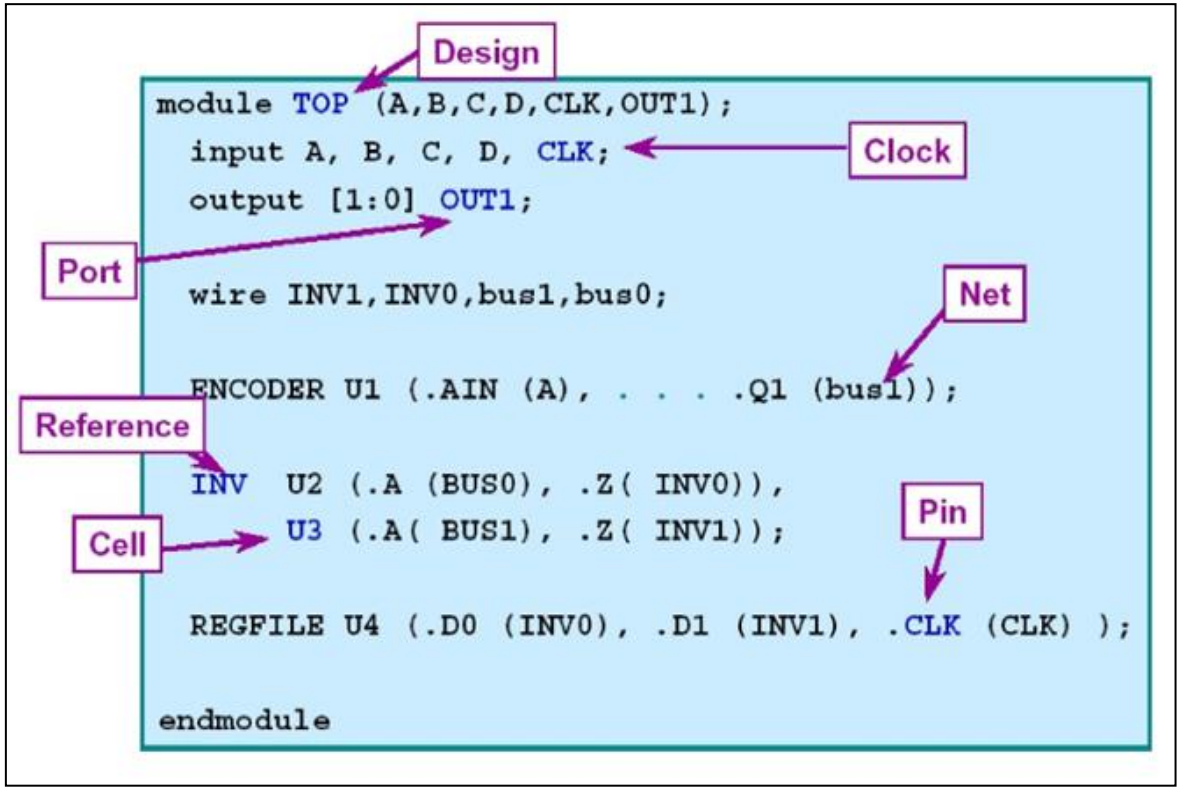
- Using Tcl with Synopsys Tools
- PrimeTime User Guide: Fundamentals:
- PrimeTime User Guide: Advanced Timing Analysis

CONTENT

TCL的应用：Design Object

通过对Design Object的分析，我们来了解和学习DC获取电路并进行解析的方法。





Schematic View and Code of Design Objects

综合软件当中TCL的常见指令

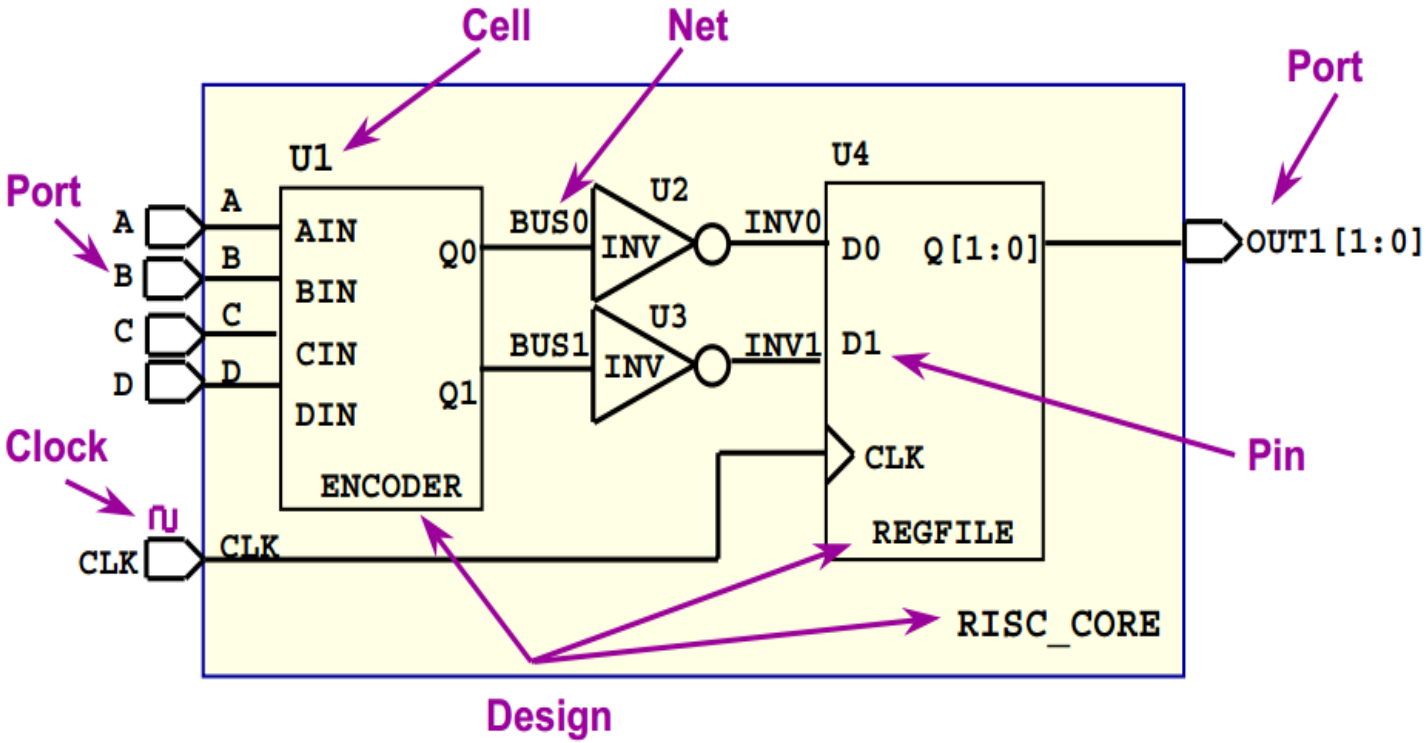
语法格式：get_ports portsName

指令功能：返回design中对应的ports object

例-1：如何查看design当中有没有一个port叫做CLK?

Shell> get_ports CLK

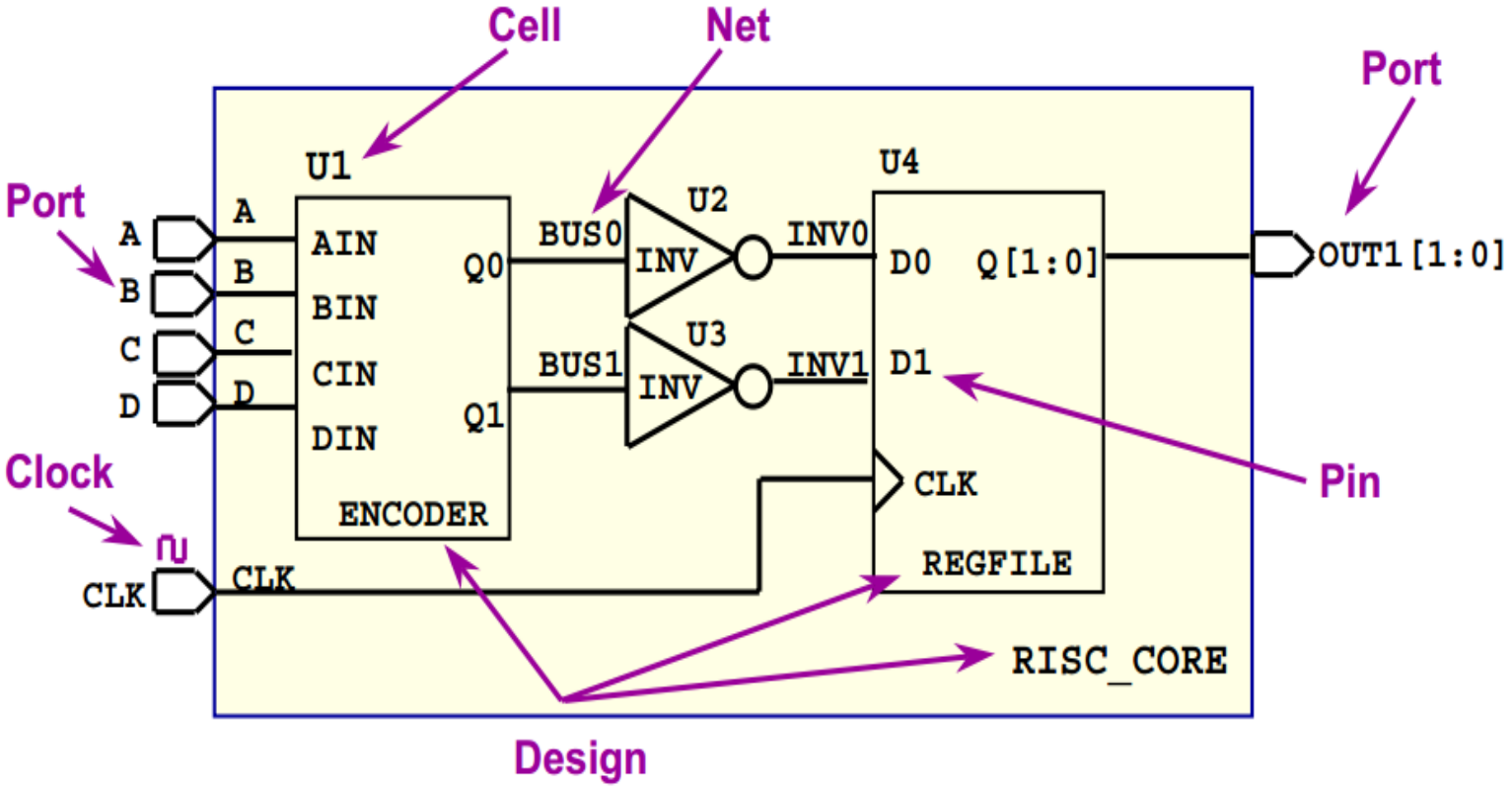
{CLK}



综合软件当中TCL的常见指令

例-2：我们想查看design当中有没有一个port叫做SPI?
Shell> get_ports SPI
No object Found!

例-3：我们想查看design当中所有的port （*可以通配任何字符）
Shell> get_ports *
{A B C D CLK OUT[0] OUT[1]}



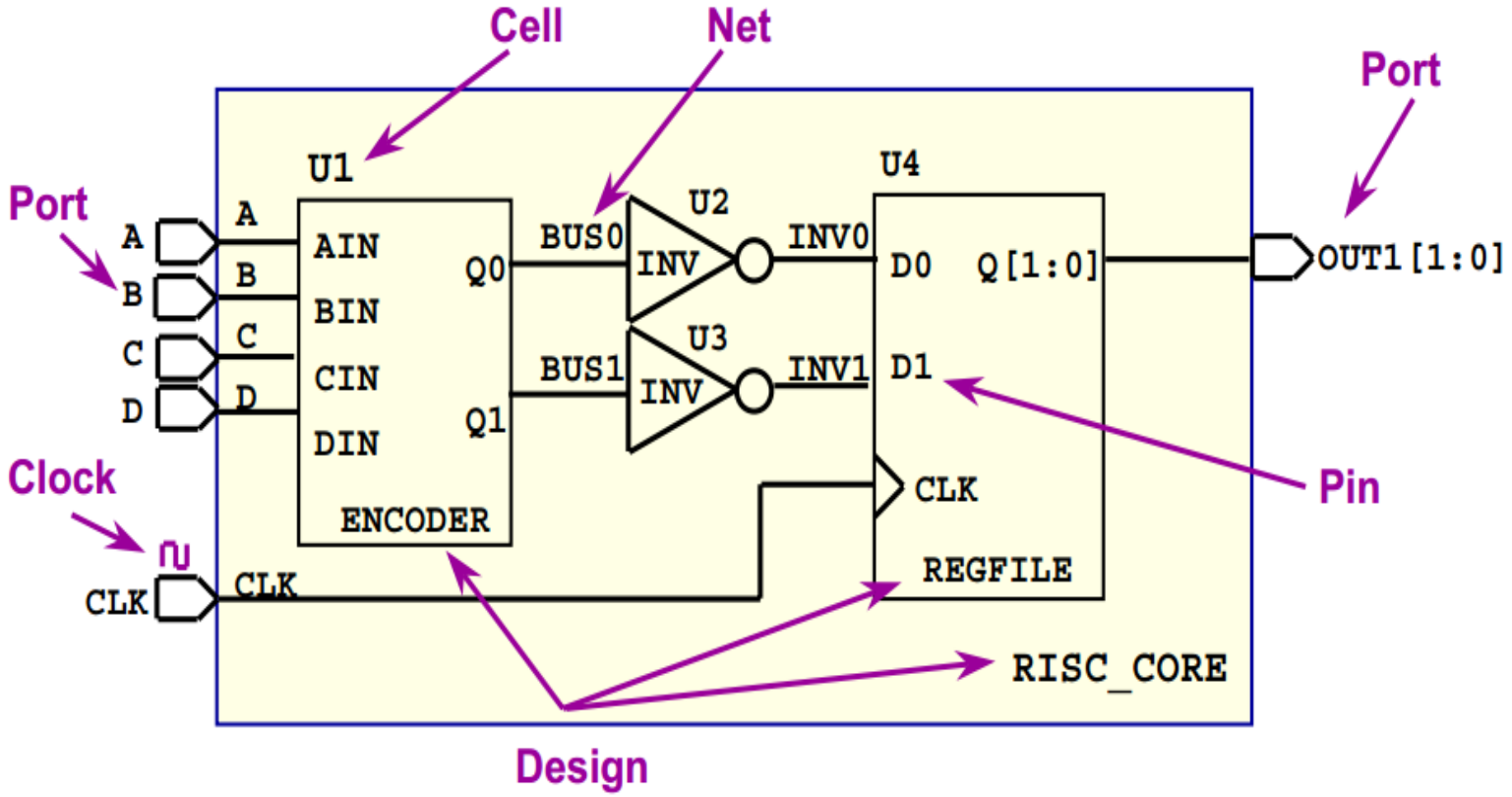
综合软件当中TCL的常见指令

例-4：假设我们有port名字叫 {CLKA CLKB OUTA OUTB INA INB}

如果我们想得到所有C开头的port 怎么做?

Shell> get_ports C*

{CLKA CLKB}



综合软件当中TCL的常见指令

语法格式：get_cells cellsName

指令功能：返回design中对应的cell的instance

name object

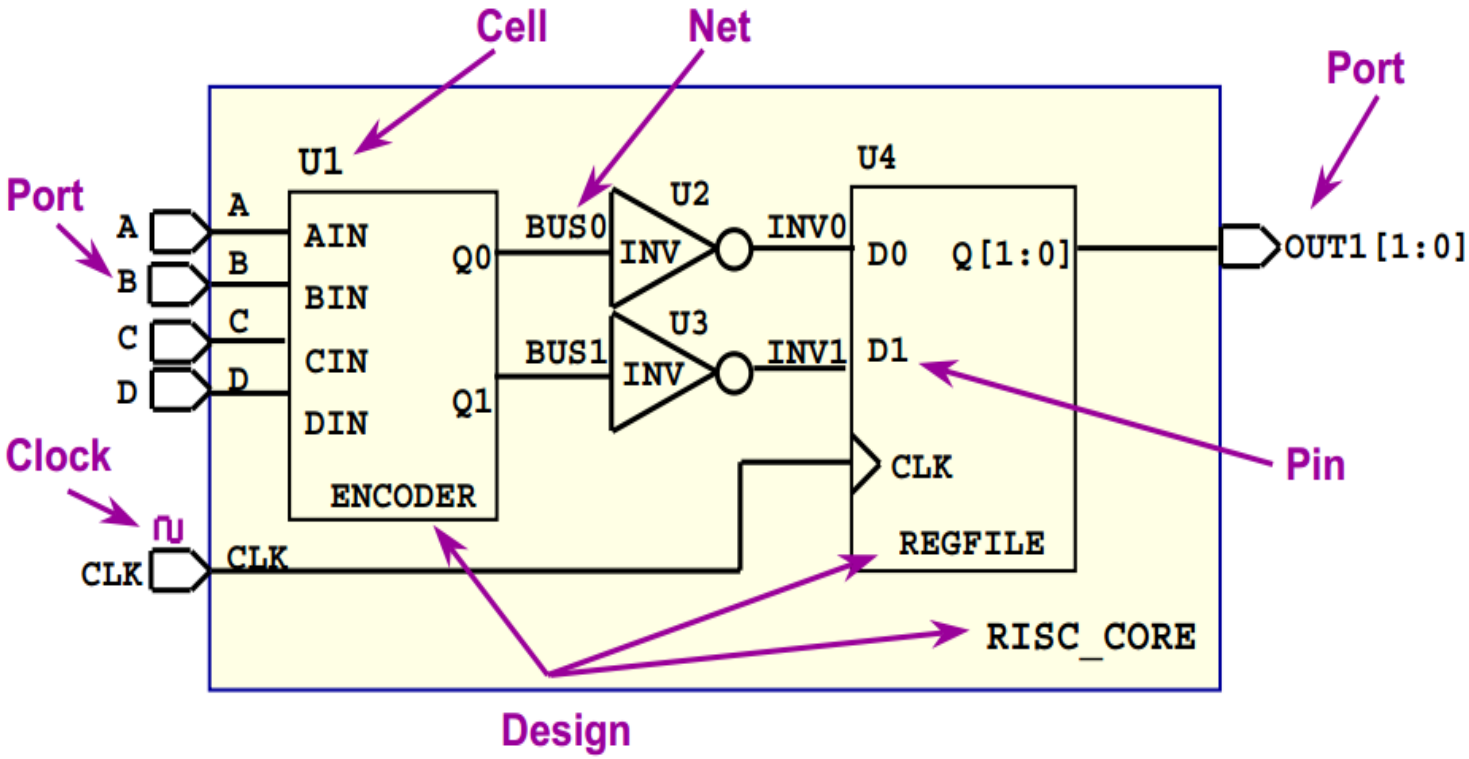
回顾几个概念：

- 什么是reference name(ref_name)?

  {ENCODER INV REGFILE}

- 什么是instance name?

  {U1 U2 U3 U4}



综合软件当中TCL的常见指令

举例-1：我们想查看design当中有没有一个cell叫做U4?

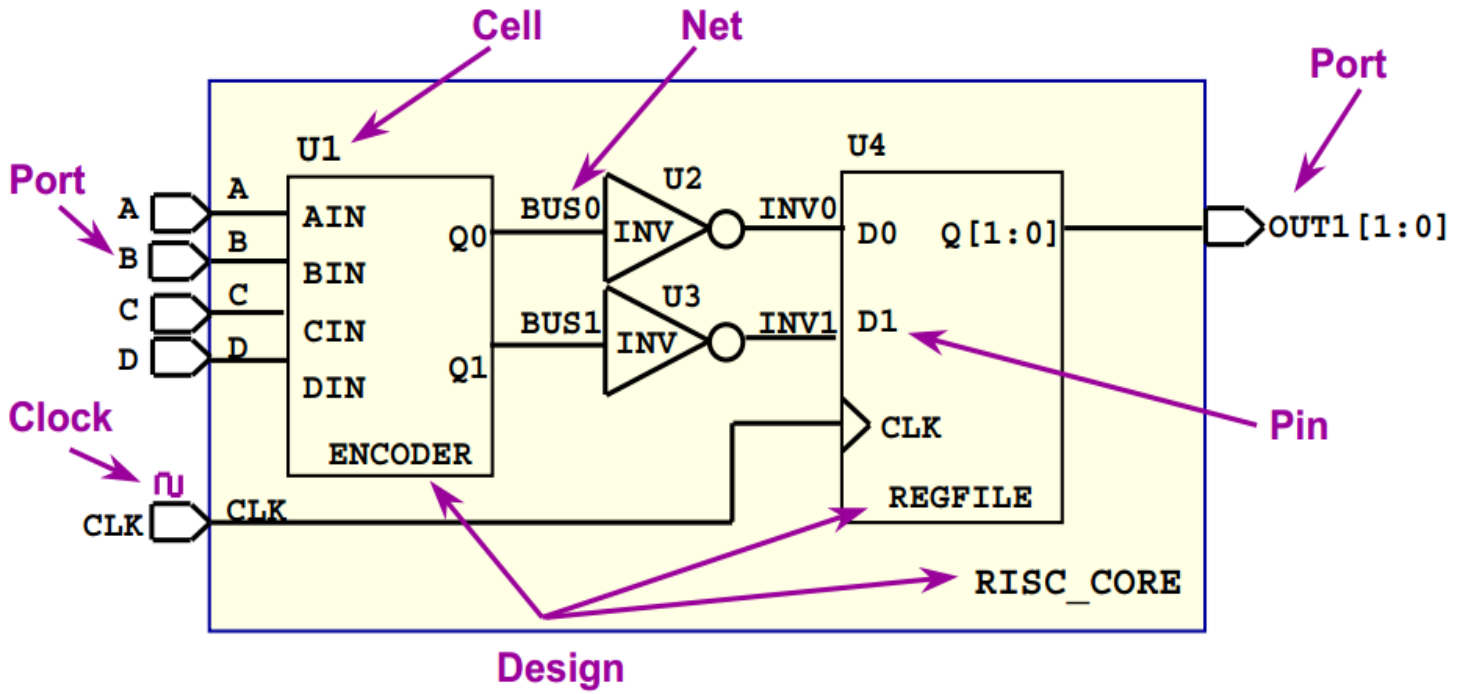
```
Shell> get_cells U4
{U4}
```

举例-2：我们想查看design当中所有的cell

```
Shell> get_cells *
{U1 U2 U3 U4}
```

举例-3：我们想查看design当中以3为结尾的cells

```
Shell> get_cells *3
{U3}
```



综合软件当中TCL的常见指令

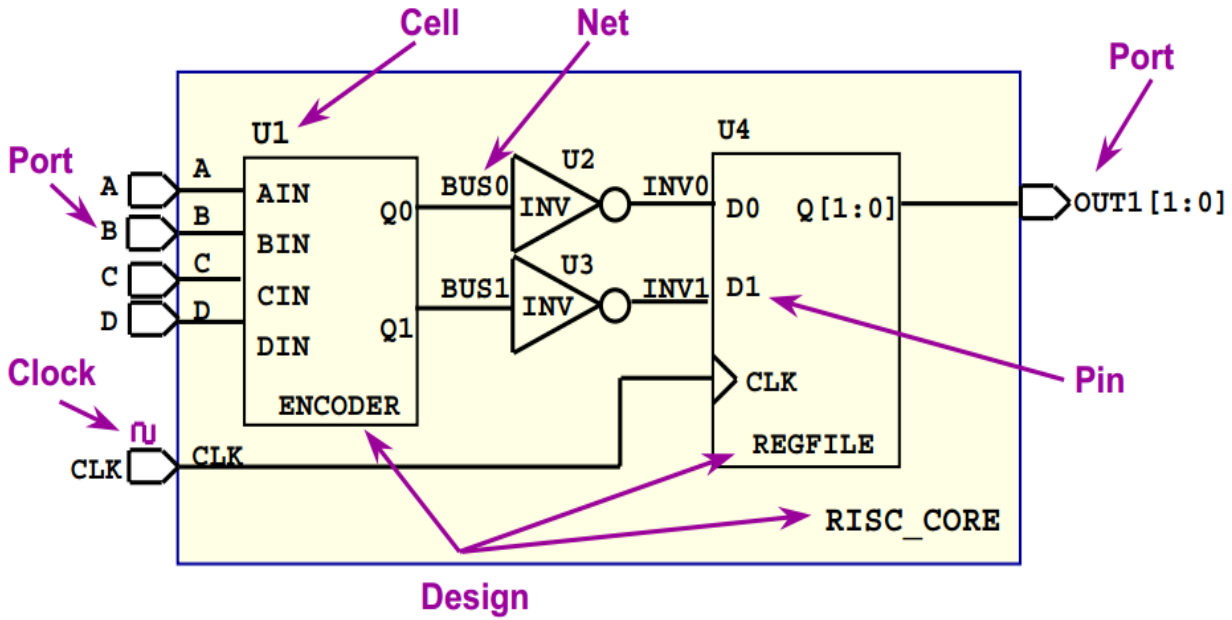
语法格式：get_nets netsName

指令功能：返回design中net的object

举例-1：查看design当中有没有一个net以INV开头?

Shell> get_nets INV*

{INV0 INV1}



综合软件当中TCL的常见指令

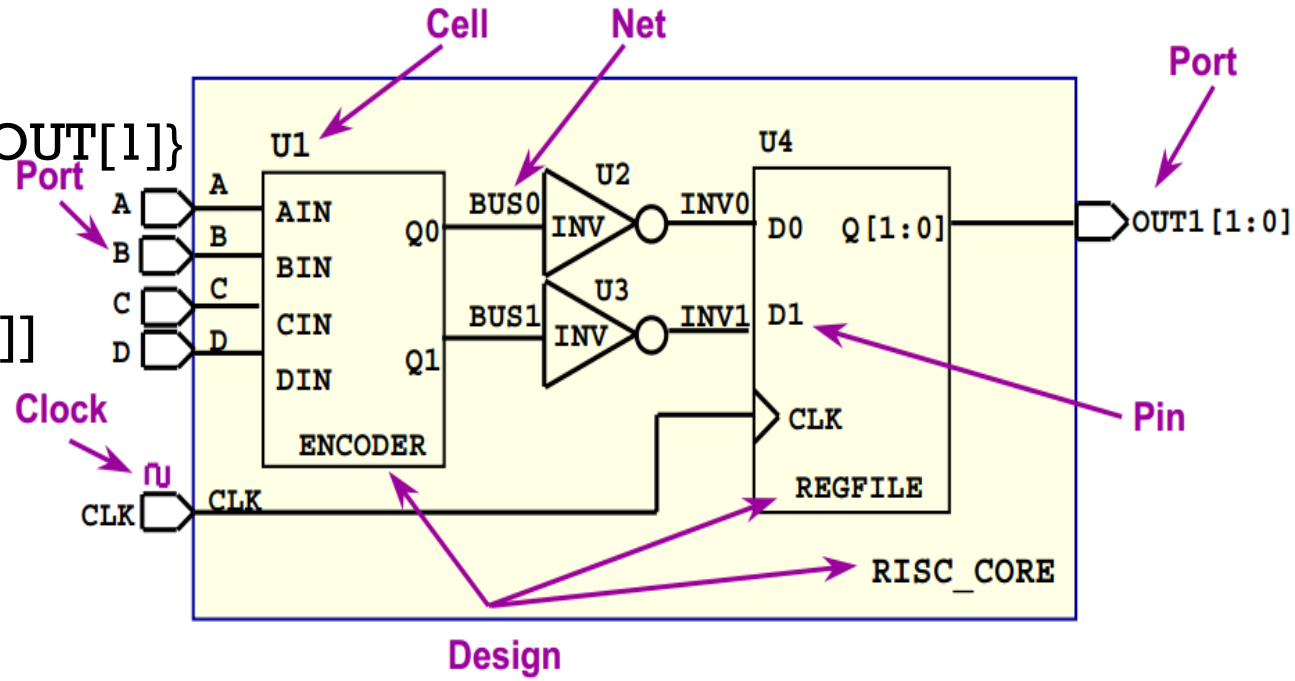
举例-2：我们想查看design当中所有的nets

```
Shell> get_nets *
{A B C D CLK BUS0 BUS1 INV0 INV1 OUT[0] OUT[1]}
```

举例-3：我们想查看design当中有多少个net?

```
Shell> llength [get_object_name [get_nets *]]
11

Shell> sizeof_collection [get_nets *]
11
```

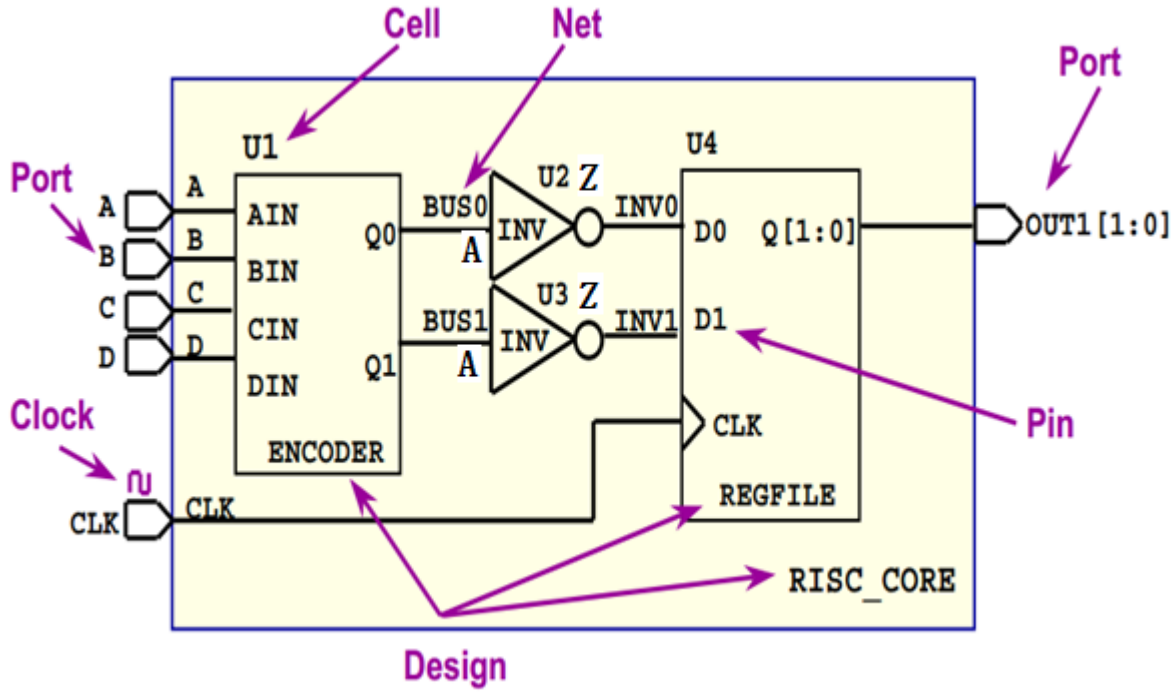
注意两条指令的区别噢！



综合软件当中TCL的常见指令

语法格式：get_pins pinsName

指令功能：返回design中pin的object



举例-1：查看design当中有哪些pin的名字叫做Z?

Shell> get_pins */Z

{INV0/Z INV1/Z}

举例-2：查看design当中有哪些pin的名字以Q开头?

Shell> get_pins */Q*

 {ENCODER/Q0 ENCODER/Q1 REGFILE/Q[1] REGFILE/Q[0]}

综合软件当中TCL的常见指令

“数据类型：object （对象）”与其“属性”

说明：

- object是对于tcl脚本一个重要的扩展；
- 常见的对象有四种 cell, net, port, pin；
- 每种object有它的属性。

下面的ppt将介绍一些常见属性。

- 任何一个属性都可以用get_attribute得到，
- list_attribute –class * 可以得到所有object 的属性，
- 部分属性可以用set_attribute来设置。

综合软件当中TCL的常见指令

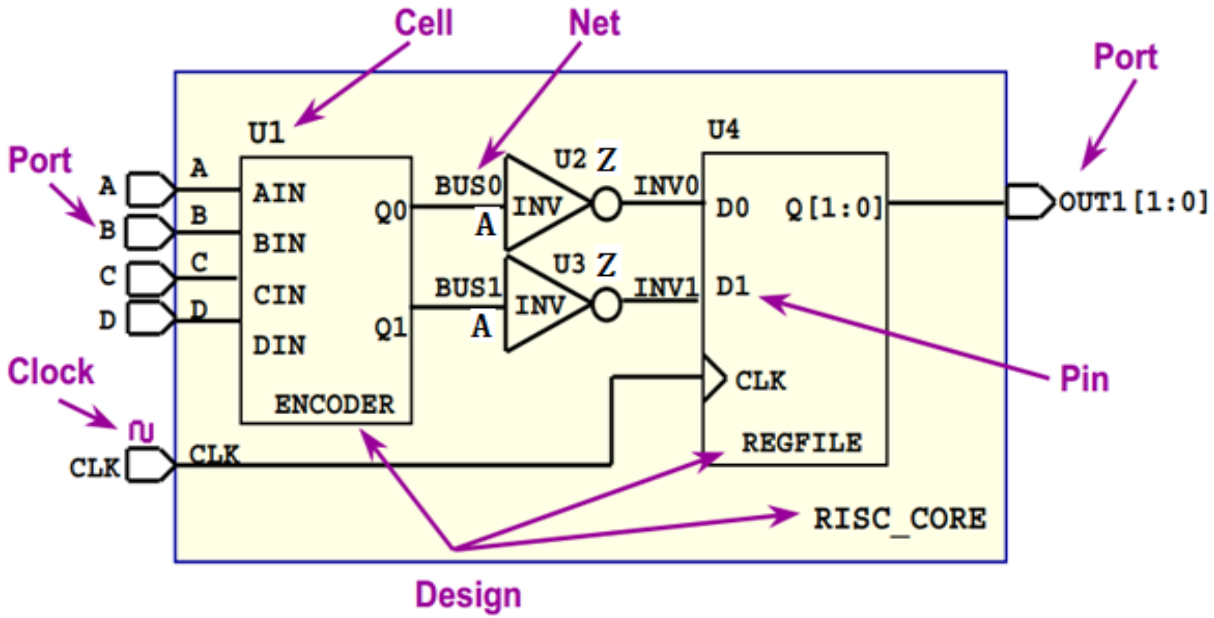
“数据类型：object （对象）”与其“属性”

- Cell object

属性 ref_name： 用来保存其map到的reference cell名称

Shell> get_attribute [get_cells –h U3] ref_name

{INV}



- Pin object:

属性 owner_net : 用来保存与之相连的net的名称

Shell> get_attribute [get_pins U2/A] owner_net

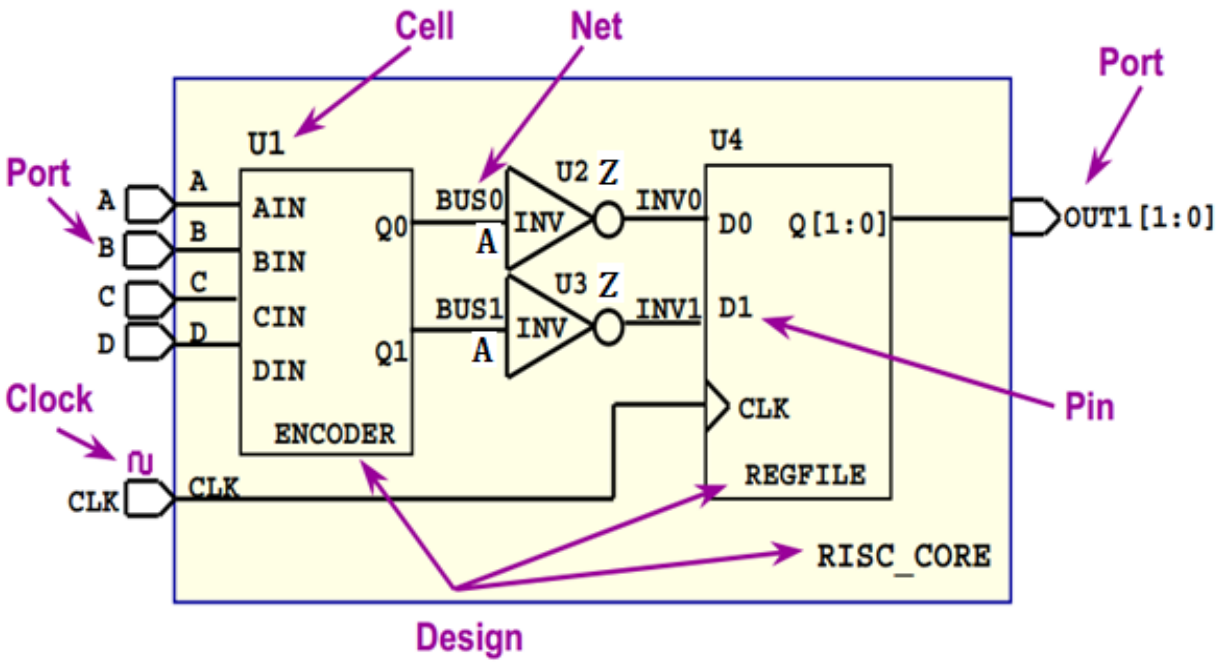
{BUS0}

综合软件当中TCL的常见指令

- Port object：

属性 direction： 用来保存port 的方向

Shell> get_attribute [get_ports A] direction
{in}
Shell> get_attribute [get_ports OUT[1]] direction
{out}



- Net object:

属性 full_name : 用来保存net的名称

Shell> get_attribute [get_nets INV0] full_name
{INV0}

Shell> get_object_name [get_nets INV0]
{INV0}

*Shell> get_attribute INV0 full_name
Error: No attribute found

综合软件当中TCL的常见指令

理解了属性，就能做更多的事情啦：

get_* -f :

-f 这个option可以用来过滤属性，以得到我们想要的object

例子-1：

想得到所有方向是input的port

Shell> get_ports * –f “direction==in”

{A B C D CLK}

例子-2：

想得到所有方向是output的pin

Shell>get_pins * -f “direction ==out”

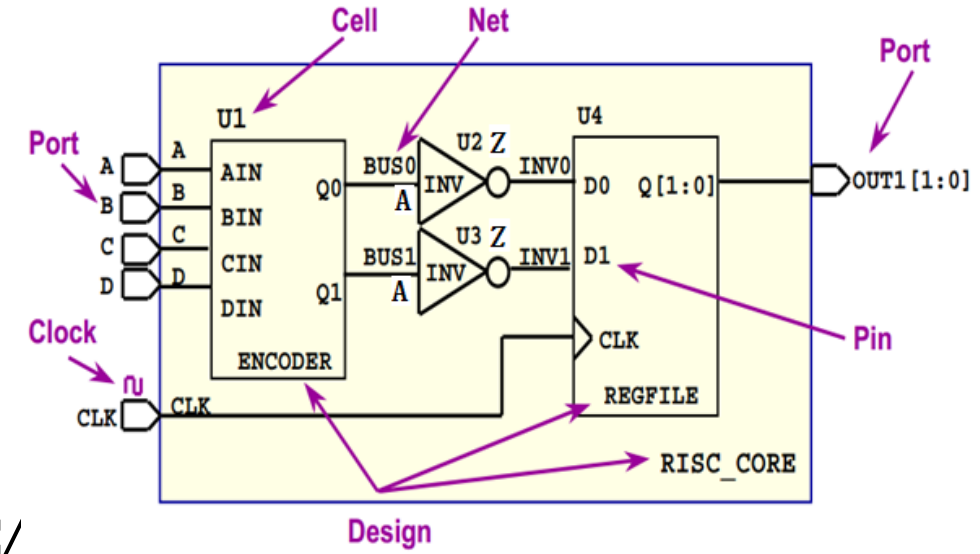
{U1/Q0 U1/Q1 U2/Z U3/Z REGFILE/Q[0] REGFILE/

例子-3：

想得到所有ref_name 是INV的 cell

Shell>get_cells * -f “ref_name == INV”

{U2 U3}

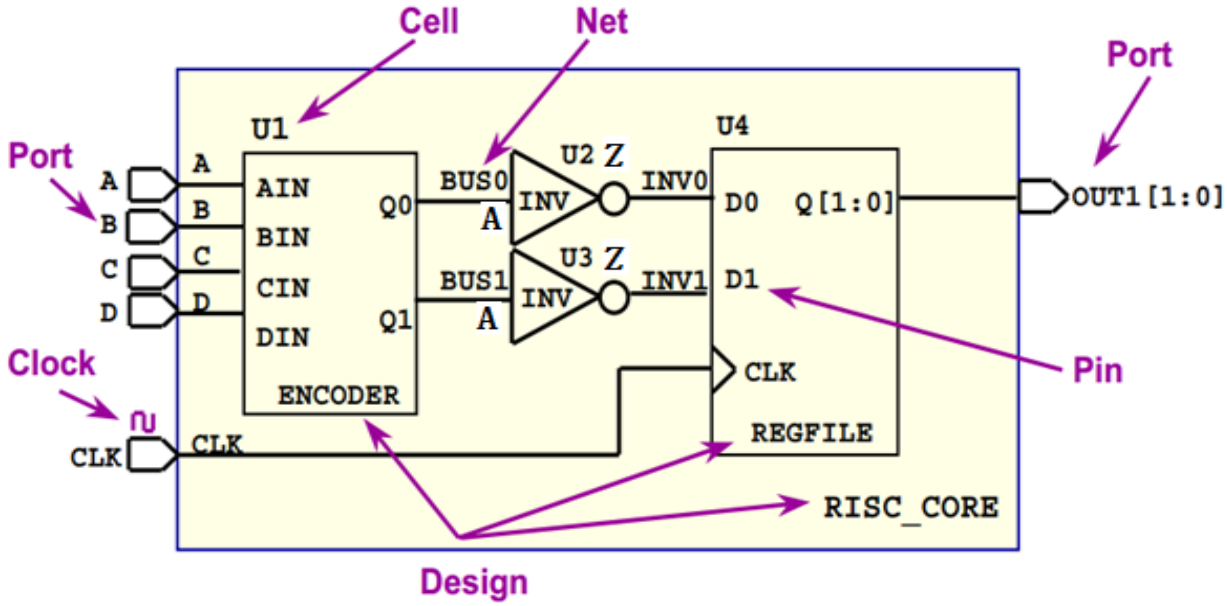


综合软件当中TCL的常见指令

另外一个最关键的option

get_* [object]-of:
-of 这个option可以用来得到与你指定object相连接的object

object的连接关系：
--port object <-> net object
>get_nets –of [get_port A]
A
--net object <-> port object / pin object
> get_net –of [get_pin U2/A]
BUS0
--pin object <-> net object
> get_pin -of [get_net INV1]
U3/Z
--cell object <-> pin object
>get_pins –of [get_cell U4]
{U4/D0 U4/D1 REGFILE/Q1 REGFILE/Q2}



综合软件当中TCL的常见指令

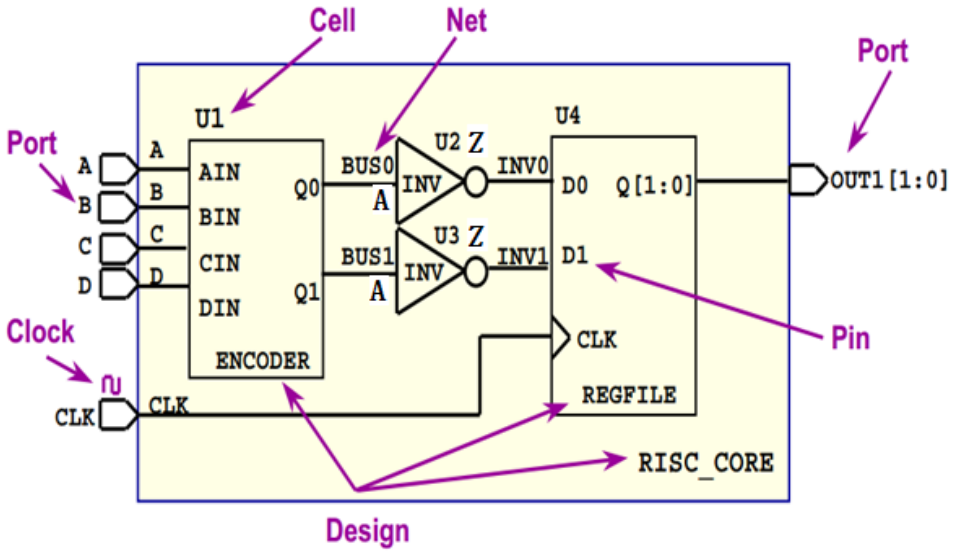
练习：

假定我们想得到电路中所有的inverter，那么应该输入什么样的脚本呢?

解答：

（1）直接使用Synopsys TCL指令：get_buffers -inverter

（2）人工扩展TCL指令：get_inverters



综合软件当中TCL的常见指令

人工扩展TCL指令：get_inverters

解题思路：

（1）使用get_lib_cells 得到所有引脚总数为2的cell

```
set lcells [filter_collection [get_lib_cells -quiet [get_attr $lib extended_name]/*] "number_of_pins==2 &&
base_name=~${pattern}"]
```

（2）使用foreach_in_collection，循环检测每一个cell。

（3）对于每一个cell，使用-filter 得到输入引脚、输出引脚的全名，以及功能名称。

```
set opin [get_lib_pins -quiet -of_object ${lcell} -filter "pin_direction==out"]
set ipin [get_lib_pins -quiet -of_object ${lcell} -filter "pin_direction==in"]
set opin_name [get_attribute -quiet ${opin} base_name]
set ipin_name [get_attribute -quiet ${ipin} base_name]
set cell_func [get_attribute -quiet ${opin}  function]
```

（4）输入引脚和输出引脚数量都为1，且功能为反相器，则就是我们想要的cell

（5）循环执行第（1）执行的步骤，最终可获得所有的反相器

CONTENT

使用TCL语言设计DC的自动化Flow

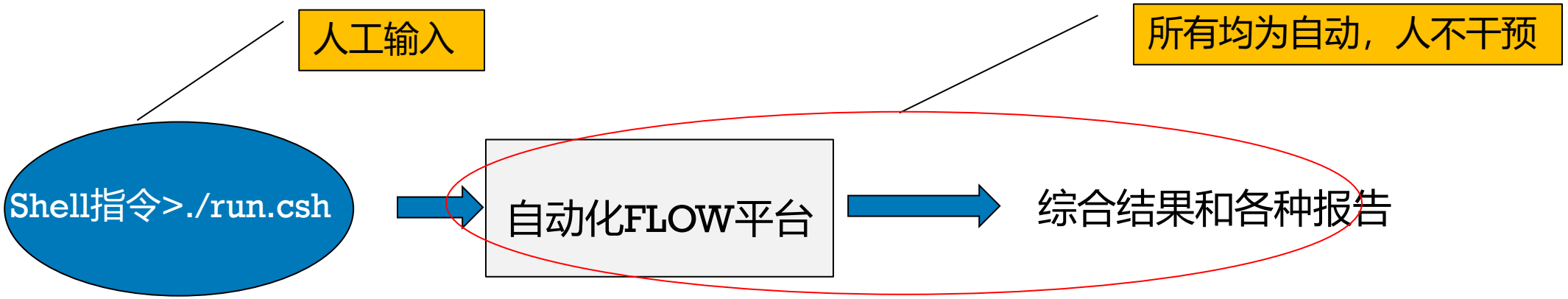
讲述一个简单的、入门级的Synopsys DesignComplier自动化方案，供初学者参考，感兴趣的同学对该方案进行优化和升级。

特点：

- 该过程无需人为操作，用户只需要输入一条shell启动指令就能完全自动化的完成整个综合过程。
- 具备普适性和可重用性。在综合不同的设计时，只需要修改参数配置文件中的环境变量，不需要修改脚本。

使用TCL语言设计DC的自动化Flow

自动化电路综合平台只需用户将待综合的设计和库文件放入一个文件夹，修改与待综合设计对象有关的环境变量参数，在命令窗口唤醒DC的指令，即自动化地完成综合过程，并得到综合后的网表文件和所需的报告。

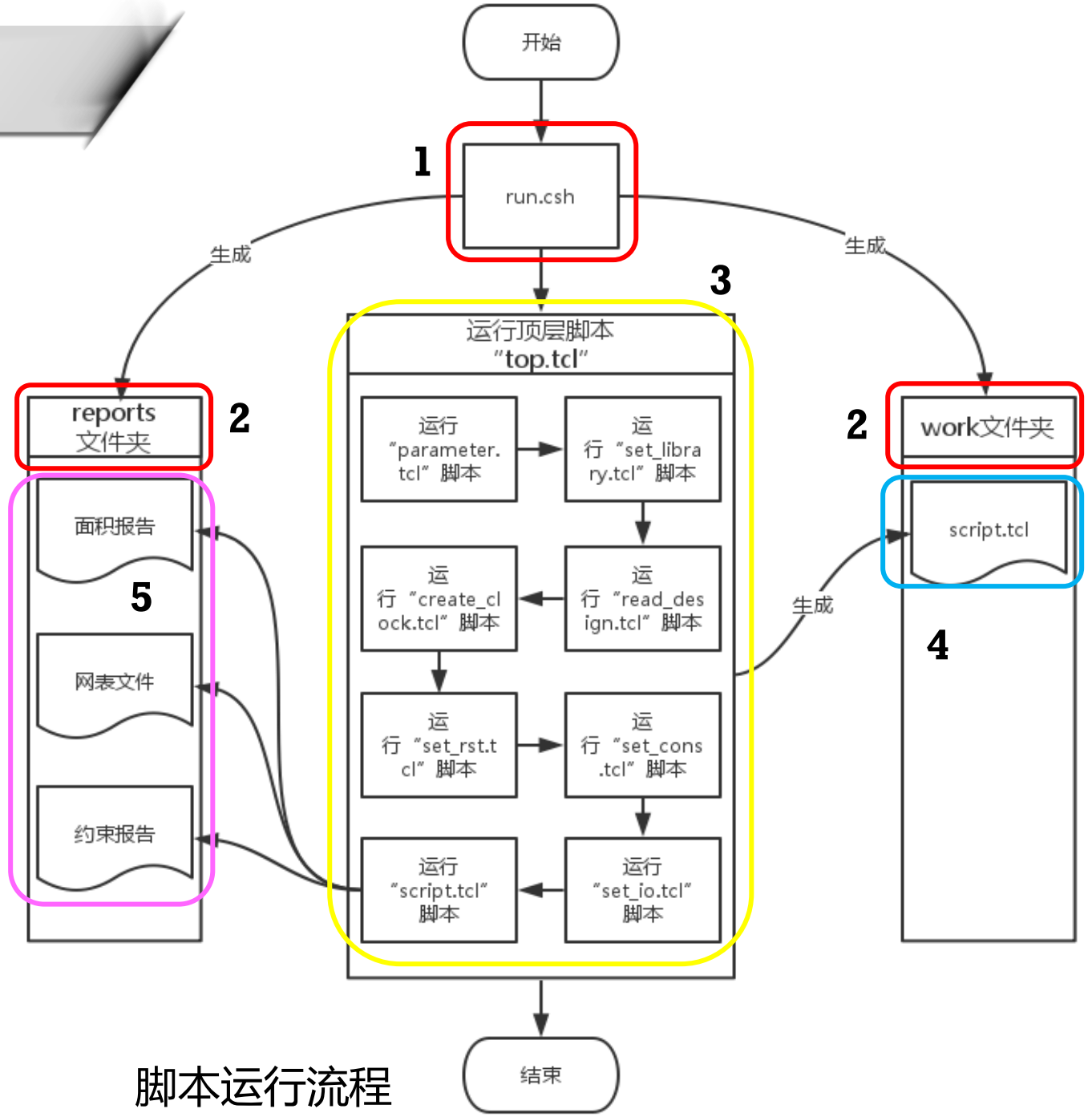


使用TCL语言设计DC的自动化Flow

运行流程：

① 运行run.csh脚本，启动DC

```
#!/bin/csh -f
\rm -rf *.log *.svf alib* reports log work   #清除之前的文档
mkdir reports work                  #新建文件夹
 dc_shell-xg-t -32bit -f ./top.tcl          #启动Design Compiler并运行top.tcl
```



脚本运行流程

使用TCL语言设计DC的自动化Flow

运行流程：

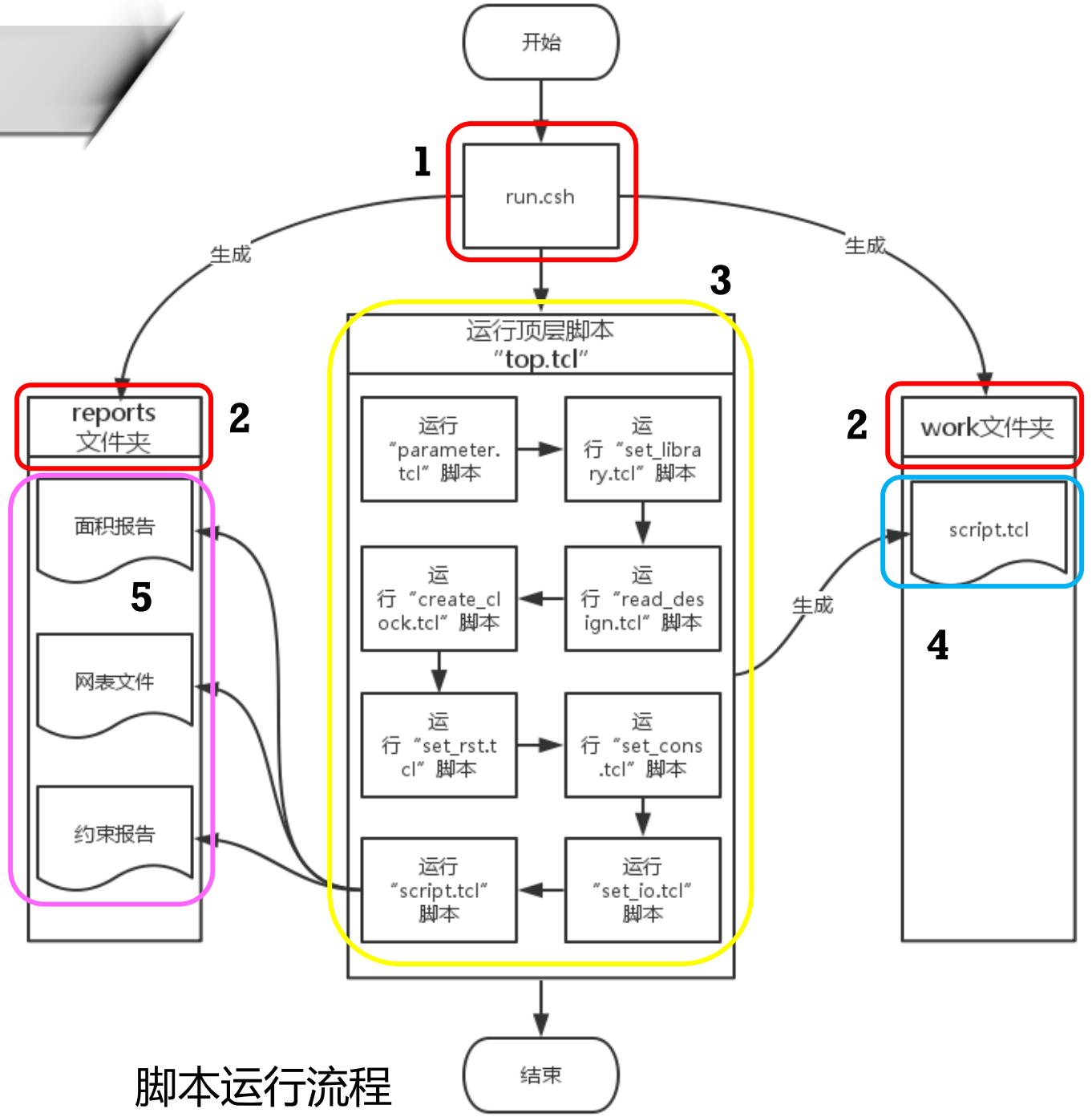
① 运行run.csh脚本，启动DC

② 建立reports与work两个文件夹。reports文件夹用于存放生成的报告，work文件夹用于存放该平台运行过程中生成的文档、脚本。

③ 启动顶层脚本top.tcl文件；

④ top.tcl按先后顺序启动各个子脚本，最终生成script.tcl；

⑤ DC读取script.tcl中的约束，最终完成综合，并将所有报告写入reports文件夹中。



脚本运行流程

使用TCL语言设计DC的自动化Flow

top.tcl脚本运行机制：

1、set_library.tcl：生成设定库文件和search path的约束

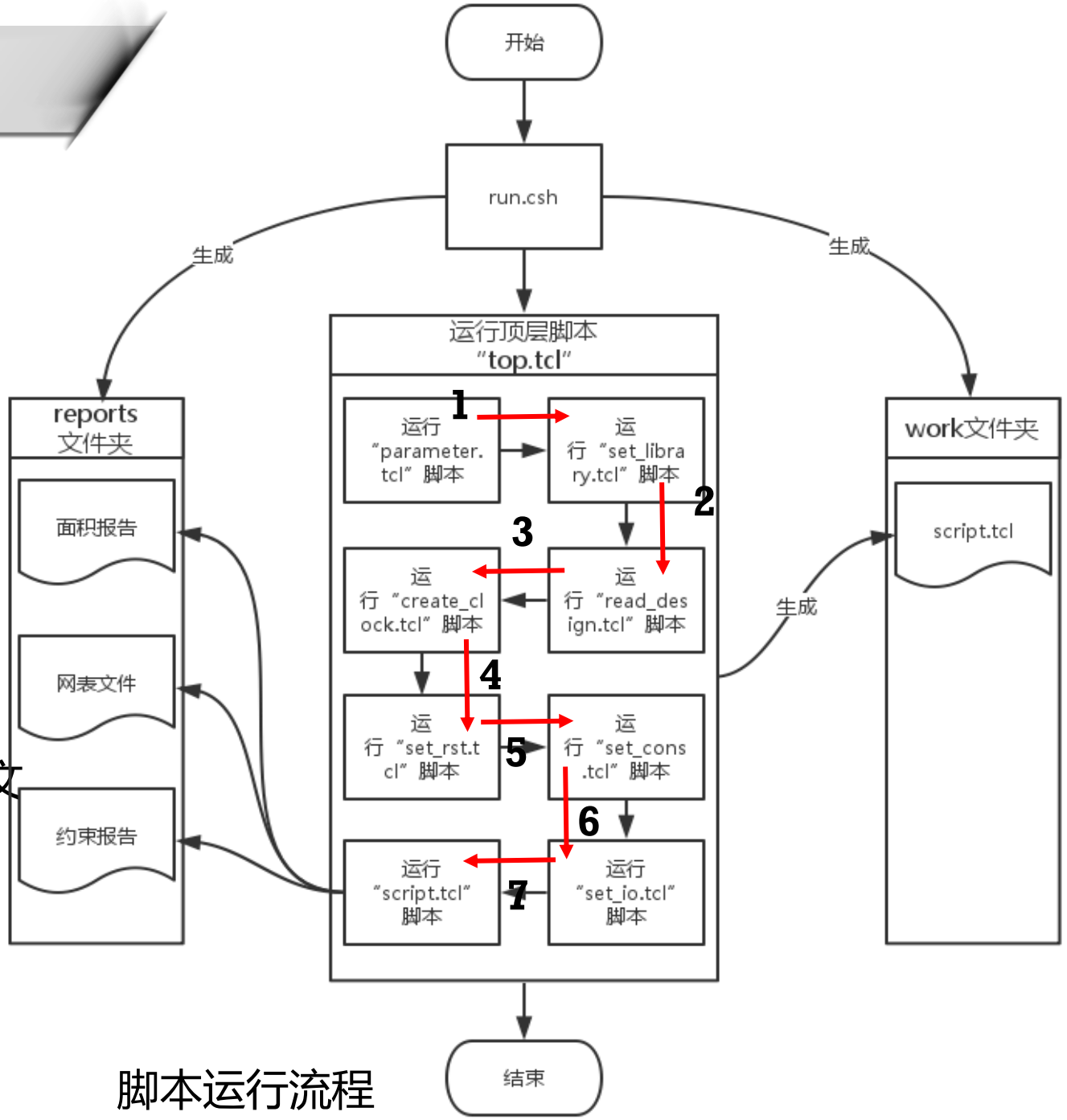
2、read_design.tcl：生成读入设计文件的约束

3、create_clock.tcl：生成时钟源相关的约束

4、set_rst.tcl：生成复位端口约束的约束

5、set_io.tcl：生成输入输出端口的约束

6、set_cons.tcl：生成保存门级网表、各种reprot文件的约束



脚本运行流程

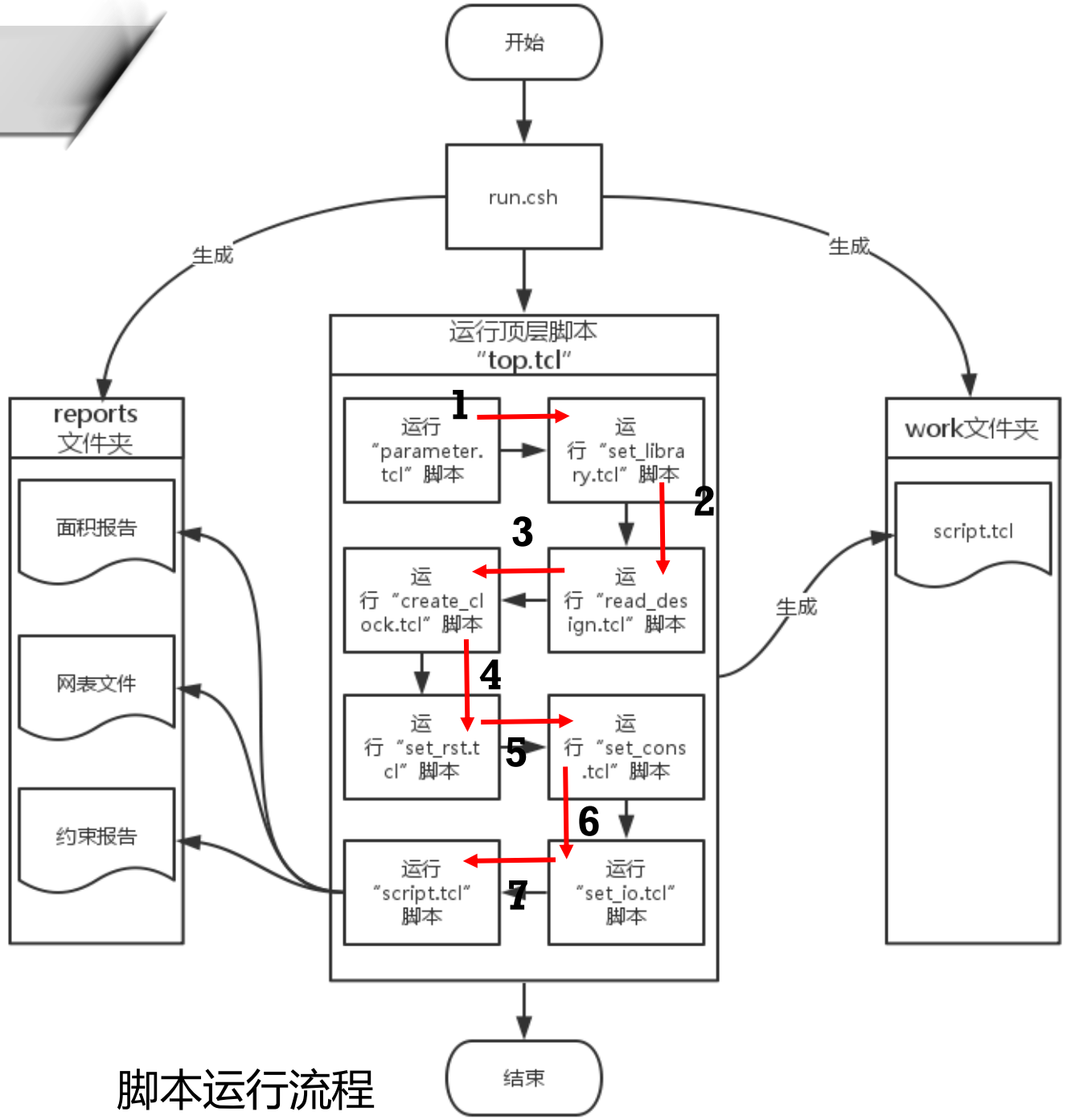
使用TCL语言设计DC的自动化Flow

注意：

1、以上所有约束都自动生成，无需人为干预；

2、库文件、代码、时钟、复位、输入输出等均根据代码自动进行匹配，并生成相应的约束；

3、最终所有的约束都被写入script.tcl中，供DC读取，完成最终的逻辑综合过程。



脚本运行流程

使用TCL语言设计DC的自动化Flow

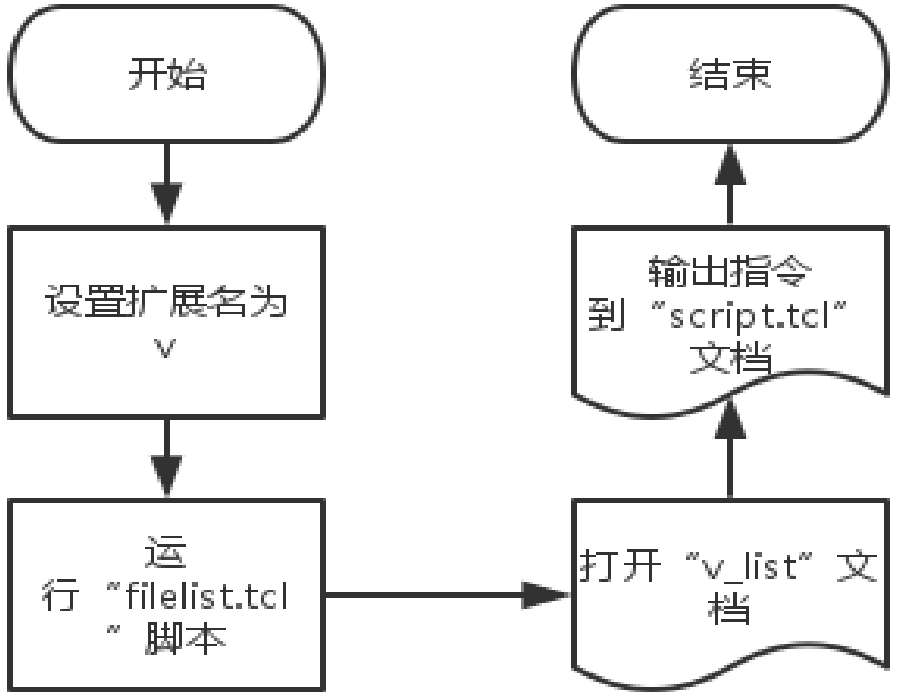
- 重要的脚本子模块——代码读入约束生成

工作流程：

① 设计文件通常使用verilog语言，所以扩展名通常为v，所以先将变量extension设为v；

② 调用filelist.tcl脚本，filelist.tcl脚本会将所有扩展名为v的文件的文件路径输出到一个名为v_list的文档。

③ 打开v_list文档，根据文件的内容将读入设计文件的指令输出到script.tcl脚本。

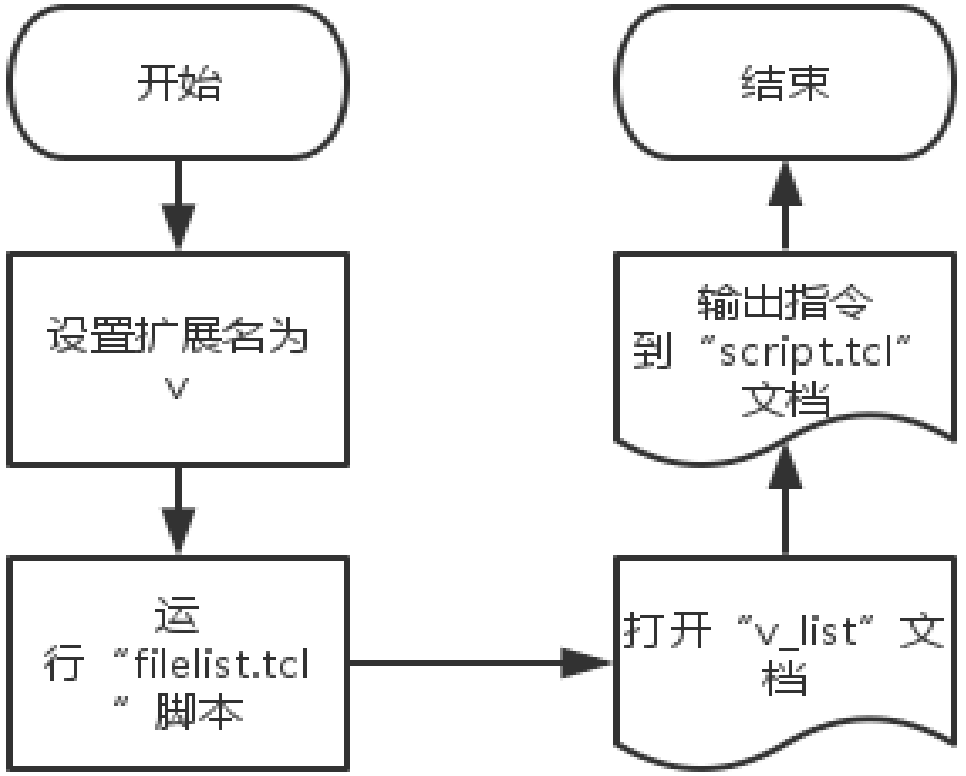


脚本运行流程

使用TCL语言设计DC的自动化Flow

- 重要的脚本子模块——代码读入约束生成

```
set extension v
source [file join $::script_path test/filelist.tcl]
                    #调用filelist.tcl脚本
set des [open [file join $::script_path test/work/v_list] r]
set design [gets $des]      #打开v_list文档
for {} {$design!=""} {set design [gets $des]} {
    puts $script [format "read_file -format verilog %s" $design]
                    #输出读入设计文件指令
}
puts $script [format "current_design %s" $top]
                    #输出设置顶层设计指令
```



脚本运行流程

使用TCL语言设计DC的自动化Flow

- 重要的脚本子模块——代码filelist生成模块filelist.tcl

功能：

本模块用filelist.tcl脚本实现，遍历目标文件下的所有文件，并将扩展名（.v）符合要求的文件完整路径输出到指定的文档，最终形成DC读取verilog代码的约束，写入script.tcl中。

使用TCL语言设计DC的自动化Flow

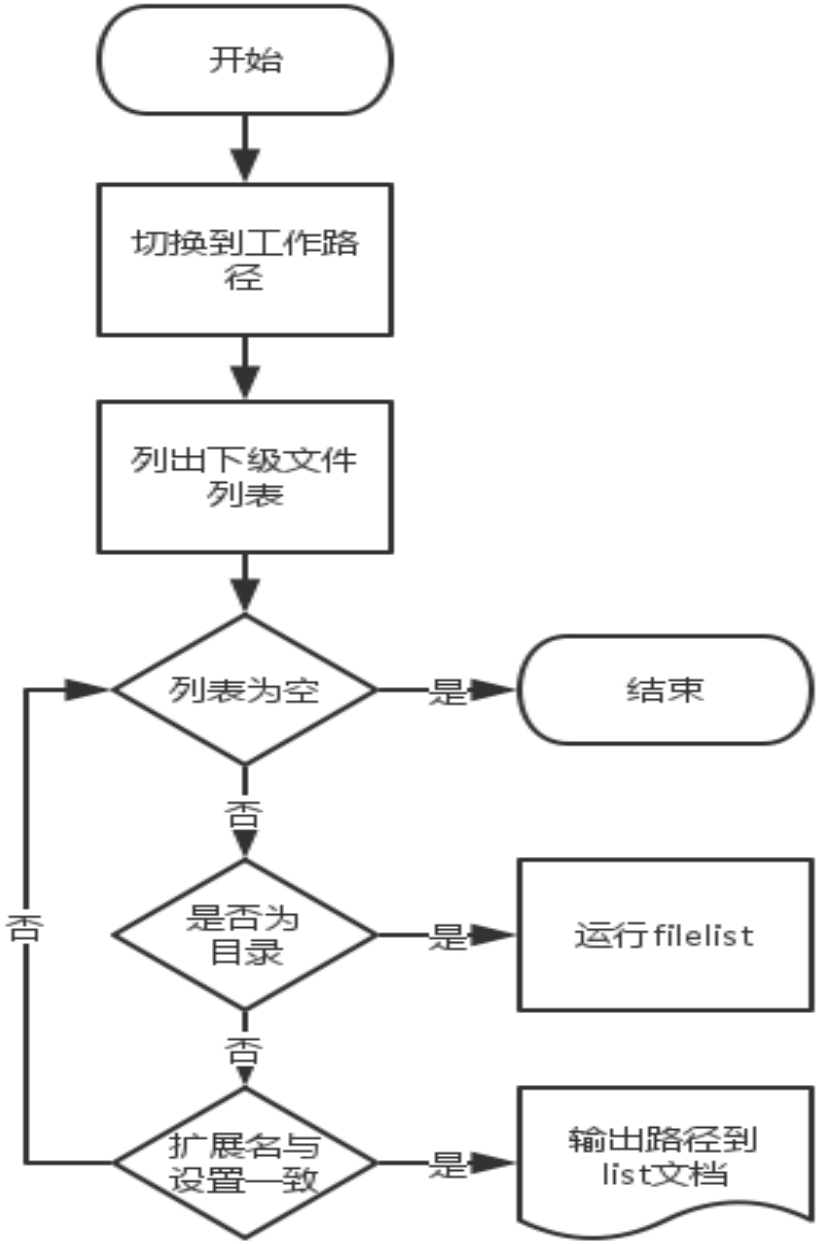
- 重要的脚本子模块——代码filelist生成模块filelist.tcl

工作流程：

① 将工作路径切换到指定的工作路径

② 判断当前目录下的文件扩展名是否与设置的变量extension一致，如果一致，就将该文件路径输出到指定的文件

③ 如果有文件夹，则递归调用本程序，直至结束。

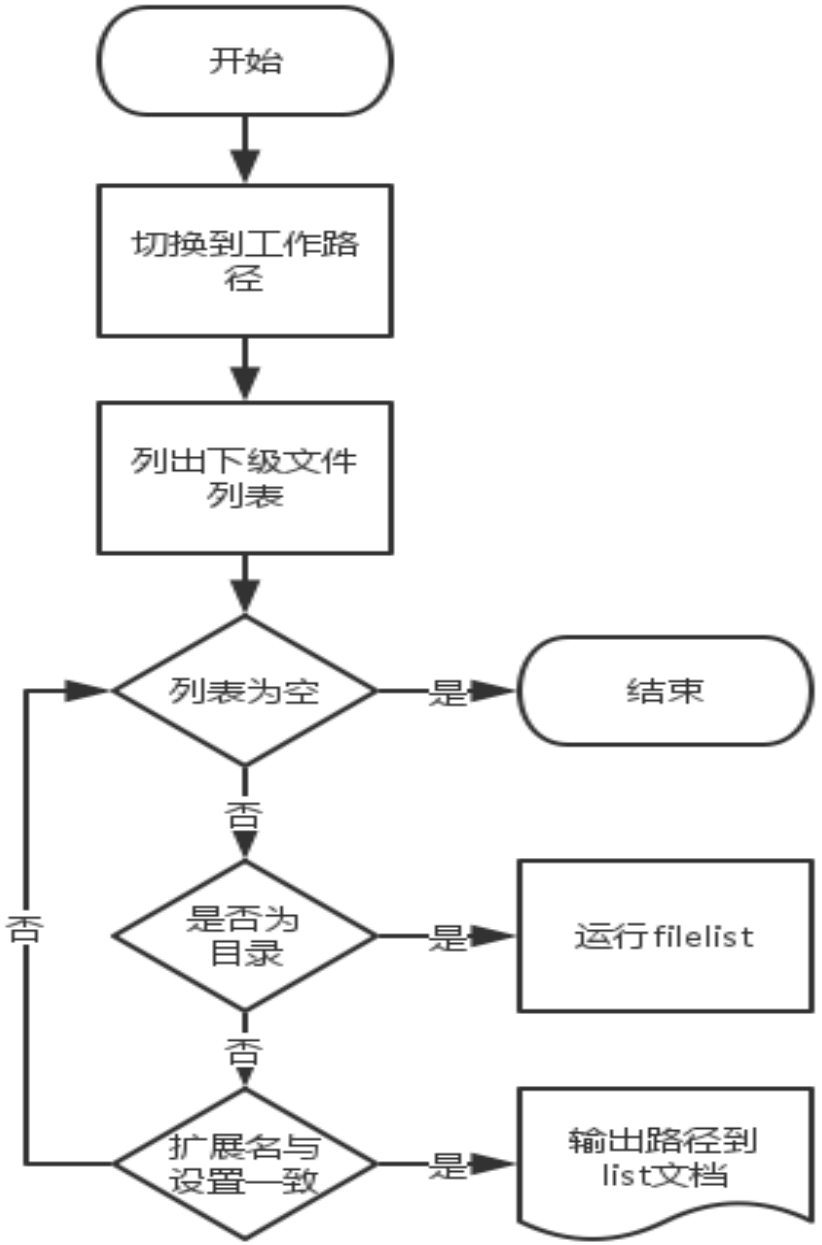


脚本运行流程

使用TCL语言设计DC的自动化Flow

- 重要的脚本子模块——代码filelist生成模块filelist.tcl

```
proc FindFile { myDir result } {
  if {[catch {cd $myDir} err]} {
    puts $result $err
    return}
  foreach myfile [glob -nocomplain *] {
     cd $myDir              #切换到对应路径
    if {[string equal $myfile ""]} {
     return }              #如果是空文件夹就返回
 set fullfile [file join $myDir $myfile]
    if {[file isdirectory $myfile]} {
     FindFile $fullfile $result      #如果有下一级路径则递归调用本函
数
    } elseif {[string equal [file extension $fullfile] [format ".%s"
$::extension]]} {              #判断扩展名是否与要求一致
     puts $result $fullfile}}}
```



脚本运行流程

使用TCL语言设计DC的自动化Flow

- 重要的脚本子模块——时钟约束生成子模块

功能和流程：

① 首先调用parameter.tcl脚本，读取其中用户对时钟源指定的参数，如时钟周期等；

② 调用find_clk.tcl脚本，该脚本会将搜索顶层设计中的所有的clk端口，并将所有搜索结果输出到一个名为clk_list的文档；

③ 打开clk_list文档，将对时钟端口施加约束的指令输出到script.tcl脚本。

DC的时钟相关的约束指令

```
source [file join $::script_path test/find_clk.tcl]
#调用find_clk.tcl脚本
set a [open [file join $::script_path test/work/clk_list] r]
#打开v_list文档
set b [gets $a]
set result [open [file join $::script_path test/work/script.tcl] a]
for {} {$b!=""} {set b [gets $a]} {
puts $result [format "create_clock -name \"clock\" -period %u -waveform {0 %d} { %s }" $::clk_source [expr $::clk_source/2] $b]
#将生成时钟源的指令输出到script.tcl脚本
   puts $result [format "set_dont_touch_network \[get_ports %s\]" $b]
#对时钟网络设置don't touch
puts $result [format "set_drive  0  \[get_ports %s\]" $b]
#设置时钟端口驱动为无穷大
puts $result [format "set_ideal_network   \[get_ports %s\]" $b]
#设置时钟端为理想网线
}
close $result
```

使用TCL语言设计DC的自动化Flow

- 重要的脚本子模块——匹配时钟端口子模块find_clk.tcl

回忆：v_list文档中存放的是什么？

功能：

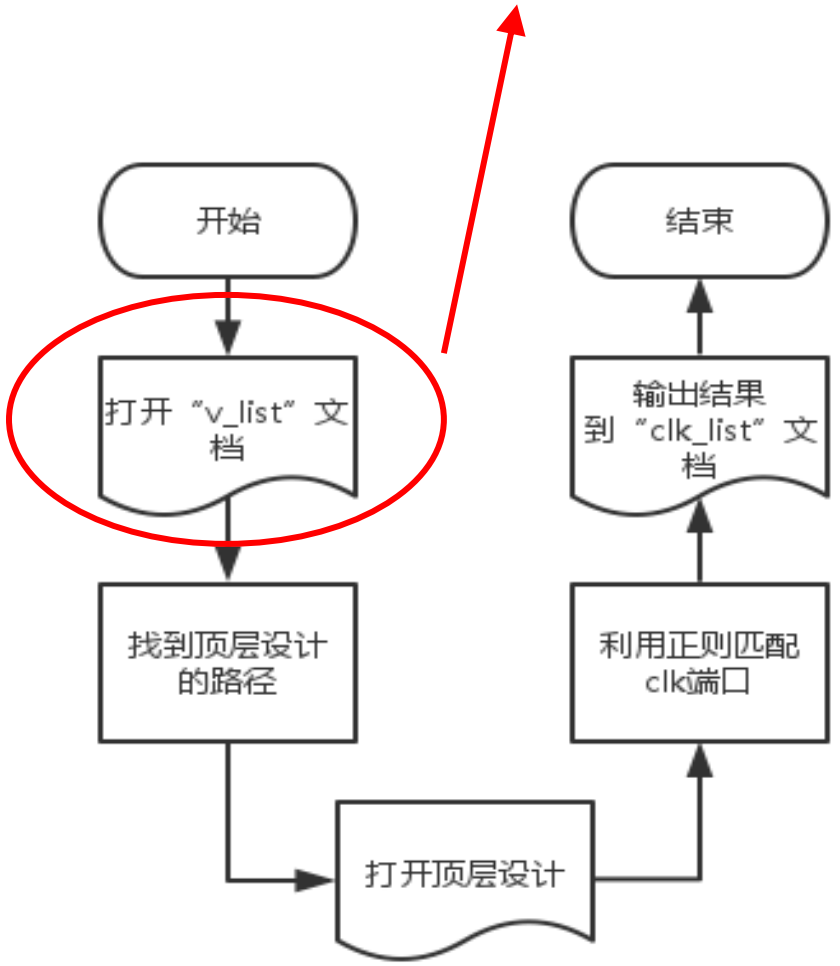
搜索代码中的所有的时钟端口，将结果到work文件夹下的clk_list文档

工作流程：

① 打开v_list文档，在其中找到顶层设计的路径，并打开改设计文件；

② 利用正则表达式匹配其中的clk端口；

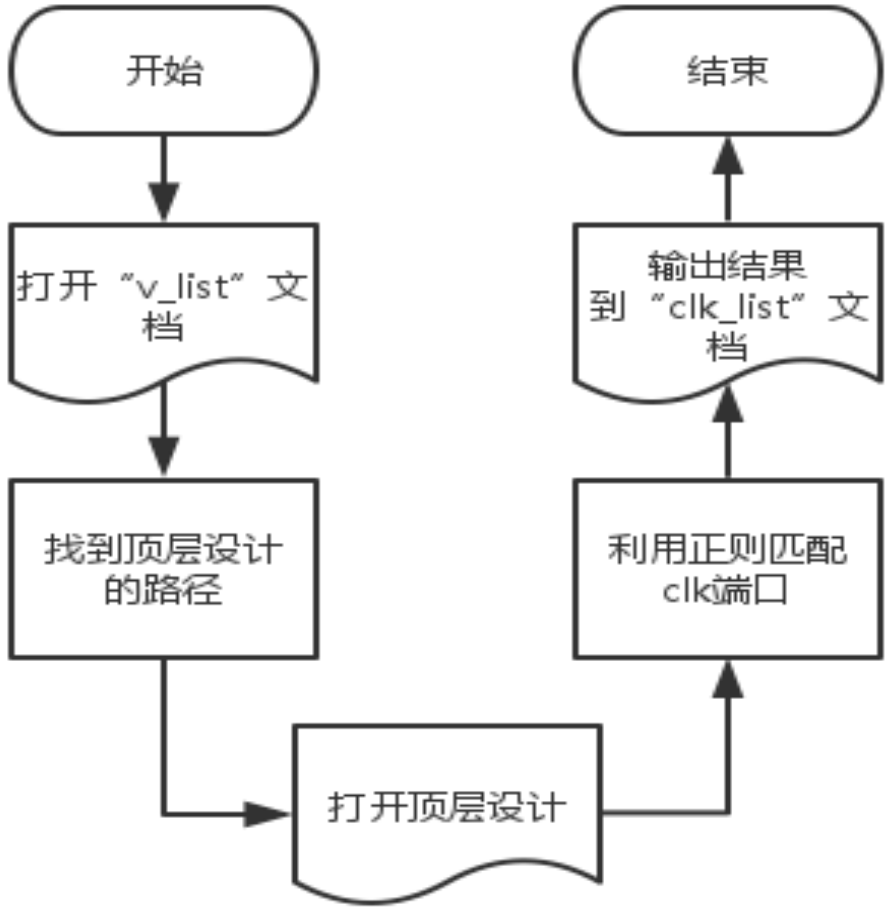
③ 并将匹配到的时钟端口的端口名输出到work文件夹下的clk_list文档。



脚本运行流程

使用TCL语言设计DC的自动化Flow

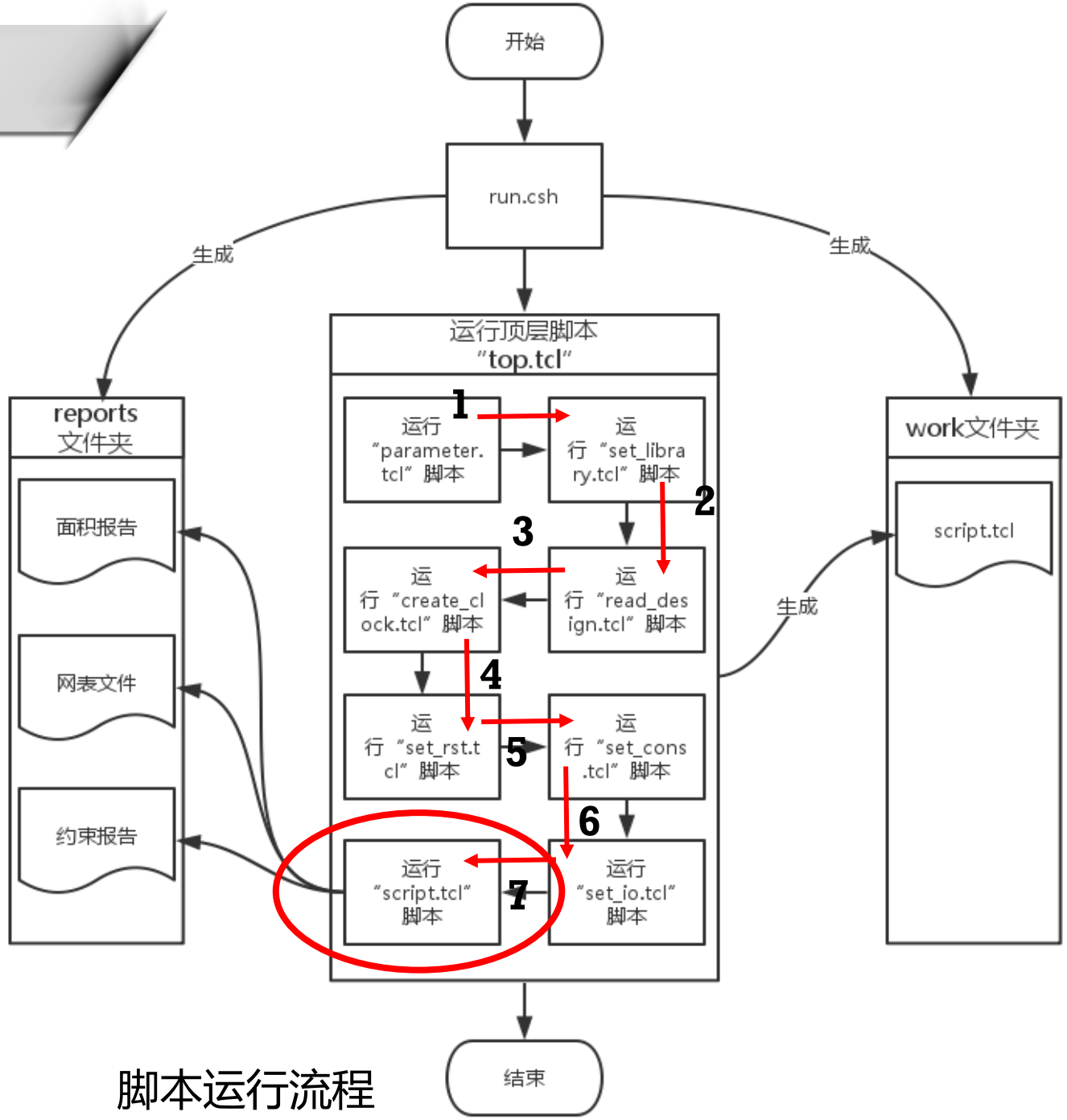
```
for {} {[eof $designfile]==0} {set fdesign [gets $designfile]} {


      if {[regexp {input.*} $fdesign a]} {
      #利用正则表达式匹配到声明输入端口那一行
          if {[regexp {[^ ,(    ]*clk[^ ,;]*} $a rport]} {
        #利用正则表达式在那一行匹配后缀为clk的端口
        puts $fport $rport
        #将匹配到的端口名输出到clk_list文档
   }
   }
}
```



脚本运行流程

使用TCL语言设计DC的自动化Flow

- 其余脚本工作机制类似。
- 最终，所有的约束都被写入了script.tcl中。
- script.tcl脚本完全由自动化电路综合平台生成，并非人工编写；
- 在自动化电路综合平台运行后，由顶层脚本调用各子模块脚本生成该脚本。生成该脚本后，由顶层脚本调用该脚本，由此实现对设计的综合。



脚本运行流程

参考书目

- Using Tcl with Synopsys Tools. Version B-2008.09, March 2011. Synopsys.
- 集成电路静态时序分析与建模. 刘峰, 机械工业出版社.出版时间：2016-07-01.

谢谢聆听！

个人教学工作主页https://customizablecomputinglab.github.io/