深圳市华为技术有限公司 研究管理部文档中心	文档编号	产品版本	密级
		1.0	内部公开
	产品名称：		共22页

Synplify快速入门

(仅供内部使用)

文 档 作 者：	牛风举	日期：	2000/ 08 /20
项 目 经 理：		日期：	/ /
研 究 部：		日期：	/ /
总 体 组：		日期：	/ /
文档管理员：		日期：	/ /

深圳市华为技术有限公司

修订记录

日期	修订版本	描述	作者
2000/08/20	1.00	初稿完成	牛风举
2000/9/20	1.10	第一次修订	牛风举

目 录

Synplify 快速入门

关键词：Synplify synthesis

摘要： 本文简单介绍了Synplify工具提供的强大功能，常用的工作界面，基本的工作流程。并在此基础上进一步深入，阐述了怎样运用批处理工作文件提高工作效率，怎样用HDL Analyst快速定位关键路径，怎样用时间约束文件使你的综合结果更加成功。

缩略语清单：

SCOPE：Synthesis Constrains Optimization Environmemt

参考资料清单				
名称	作者	编号	发布日期	查阅地点或渠道
Synplify手册	Synplicity公司			On_line help

1 前言（overview)

Synplify 是 Synplicity 公司提供的针对FPGA和CPLD实现的逻辑综合工具。

该软件提供的synbolic FSM compiler 是用来专门支持有效状态机优化的内嵌工具，SCOPE是用来管理（包括输入和查看）设计约束与属性的提供活页式分类的非常友好的表格界面，用于文本输入的编辑窗口不仅提供了对综合错误的高亮显示，还提供了强大的cross_probe功能，可以把源代码与综合的结果有机地链接起来，帮助设计者迅速定位和解决问题。

2 基本概念

2.1 综合

（Synthesis）简单地说就是将HDL代码转化为门级网表的过程。Synplify 对电路的综合包括三个步骤，表示如下：

1、HDL compilation：把HDL的描述编译成已知的结构元素。

2、Optimization：运用一些算法对你的设计进行面积优化和减小时延。在没有目标库的情况下，Synplify只执行一些最基本的优化措施。

3、Technology mapping：把你的设计映射到指定厂家的特定器件上，执行一些附加的优化措施，包括根据由器件供应商提供的专用约束进行优化。

2.2 工程

工程文件（以*.prj 作为扩展名）以tcl 的格式保留了以下信息：设计文件、约束文件、综合时开关选项的设置情况等。

2.3 Tcl scripting

Tcl （Tool Command Language）是一种非常流行的工业标准批处理描述语言，常用作软件应用的控制。

应用Synplify 的Tcl script 文件，设计者可以用批处理命令的形式执行一个综合，也可以一次执行同一设计多个综合，尝试不同的器件，不同的时延目标，不同的约束条件。

Synplify 的script 文件以（*.tcl）保存。

2.4 约束文件

约束文件以（*.sdc）保存。用来提供设计者定义的时间约束，供应商定义的属性等。

约束文件既可以被添加到在工程窗口的代码菜单中，也可以被Tcl script 文件调用。

2.5 宏库

Synplify 在它内建的宏库中提供了由供应商给出的宏模块。比如一些门电路，计数器，寄存器，I/O模块等。你可以把这些宏模块直接例化到你的设计中去。

2.6 属性包

Synplify在Synplify_install_dir/lib/vhd/synattr.vhd 提供了一个属性包。该属性包的内容有时间约束，包括对黑匣子的时间约束，供应商提供的一些属性，还有一些综合属性以帮助你实现你的综合目的。你只需在设计文件的开头加入以下属性包调用语句。

```
library synplify;
use synplify.attributes.all;
```

2.7 对HDL语言的支持

1、支持Verilog 95（IEEE 1364）。

2、支持VHDL93（IEEE 1076）。

包括std_logic_1164，Numeric_std，std_logic_Usigned，std_logic_Signed，std_logic_Arith。

3 基本流程

3.1 启动Synplify

在UNIX环境下，键入synplify。系统弹出工程窗口。在工程窗口中包含了以下内容：源文件信息、结果文件信息、目标器件信息。

3.2 添加源文件

在ADD SOURCE FILE区域添加源文件。

3.3 选择顶层设计

Synplify把最后编译的module/entity and the architecture作为顶层设计。故把你所要的顶层设计文件用左键拖拉到源文件菜单的末尾处。

3.4 选择目标器件，设置开关选项

选择Target--->Set Sevice Options即可。

3.5 综合

点击RUN即可。

3.6 保存工程文件

选择File--->Save AS。

4 批处理工作模式流程

4.1 运行工程文件

1、启动Synplify 工程窗口。

2、设置你的工程选项。

3、设置好工程文件：源代码文件，约束文件，Tcl scripts文件.

4、保存工程文件*.prj。

5、运行工程文件。

```
synplify -batch  project_file_name.prj
```

4.2 运行一个Tcl文件

1、编写一个Tcl 文件。格式如下：

```
project -new
#all your other Tcl commands will to there
project -run
exit
```

2、输入你的Tcl 命令。

3、保存你的文件。把它与你的工程文件、源代码文件、约束文件放在一起。

4、运行：

```
synplify  -batch  Tcl_script_name.tcl。
```

5 怎样用Tcl语言执行批处理任务

Tcl （Tool Command Language）是一种非常流行的工业标准批处理描述语言，常用作软件应用的控制。

应用Synplify 的Tcl script 文件，设计者可以用批处理命令的形式执行一个综合，也可以一次执行同一设计多个综合，尝试不同的器件，不同的时延目标，不同的约束条件。

Synplify 的script 文件以（*.tcl）保存。

5.1 创建Tcl script 文件

1、**建立新工程**：

project -new

2、**添加源文件**：

add_file -verilog

或 add_file -vhdl

3、**用综合控制命令去设置目标器件、设计速度目标等。调用**symbolic FSM compiler**及其他**option**设置**。

set_option

4、**用供应商提供的**vendor --specific Tcl **命令去设置目标工艺器件、封装、速度等级，还可以改变一些隐含设置如**fan-out **等**。

vendor--specific

5、**添加约束文件**。

add_file --constraint

6、**执行综合命令**

project --run

7、**保存文件为** *.tcl

说明：Tcl **文件的注释行以** # **开头**

对文件中的路径名和文件名要用双引号括起来

5.2 常用Tcl 命令说明

5.2.1 工程命令

1、project --new

创建一个新工程。该新工程的名字有project --save **命令指定**。

你必须在运行其他Tcl **命令之前运行** project --new **和** project --load **两者之一**。

2、project --load "project_name.prj"

装载一个工程文件。

3、project --log_file "new_log_filename.srr "

指定一个新的 log **文件名代替隐含的** log **文件名**project_name.srr 。**该**log**文件包含了以下信息：简单的编译信息，技术映射信息，资源利用信息和时序报告**。

4、project --result_file "result_filename "

指定一个新的综合结果文件名代替隐含的综合结果文件。

注意：**文件名要小写**。

改变文件的扩展名并不能改变综合结果文件的数据格式。

5、project --result_format "result_file_format "

改变综合结果文件的数据格式。

注意：改变文件的数据格式并不能自动改变文件的扩展名，仍需用project --result_file。

6、project --compile

编译你的设计工程而不进行技术映射。

系统执行语法检查，可综合性检查，生成RTL**级综合结果。**

7、project --run

编译并综合你的设计工程。

8、project --save "project_filename.prj "

保存工程文件。

5.2.2 添加文件的命令

1、add_file --verilog "verilog_filename.v "

添加verilog**格式的**HDL**源文件。你可以用**"*.v"**代表你所有的文件。**

2、add_file --vhdl [-lib library_name] "vhdl_filename.vhd "

添加vhdl**格式的**HDL**源文件。**

3、add_file --constraint "constraint_filename.sdc"

添加约束文件。

5.2.3 控制命令

1、set_option --top_module {verilog_module | vhdl_entity | vhdl_entity.arch}

指定顶层设计

2、set_option -write_verilg {true | false}

把综合生成的网表存成一个verilog**格式的文件，以便综合后仿真调用。**

3、set_option --write_vhdl {true | false}

把综合生成的网表存成一个vhdl**格式的文件，以便综合后仿真调用。**

4、set_option --write_apr_constraint {true | false}

把综合中的约束信息生成一个指导布局布线的约束文件，文件的格式与所选器件有关。

5、set_option --frequency MHz_frequency

指定时间约束目标。

6、set_option --technology {vendor_technology}

指定综合选用的器件系列。

7、set_option --part {vendor_part_name}

指定综合选用的器件名。

8、set_option --package {vendor_package_name}

指定综合选用器件的封装。

9、set_option --speedgrade {number}

指定综合选用器件的速度等级。

10、set_option --symbolic_fsm_compiler {true | false}

选择是否在综合时调用针对状态机优化的强有力的工具symbolic FSM compiler。

11、set_option --default_enum_encoding {onehot | sequential |gray}

指定在对设计中的状态机进行综合时选用的隐含编码方式：one-hot，sequantial，gray等。

5.2.4 打开文件的命令

1、open_file --edit_file "filename"

在Synplify内嵌的编辑窗口打开源文件或其他文本文件。

2、open_file --rtl_view

在HDL Analyst窗口中打开当前工程综合结果的RTL级显示。

3、open_file --technology_view

在HDL Analyst窗口中打开当前工程映射后的结果显示。

5.3 Tcl 格式的script文件示例

5.3.1 运行一个script 文件针对多个目标器件进行综合

```
# Run synthesis multiple times without exiting while trying different
# target technologies. View their implementations in HDL Analyst.
# Open a new Project.
        project -new

# Set the design speed goal to 33.3 MHz.
        set_option -frequency 33.3

# Add a Verilog file to the source file list.
        add_file -verilog "andorxor.v"

# Create a new Tcl variable, to be known as $try_these, that will be
# used to synthesize the design using different target technologies.

set try_these {
                ACT3
                200DX
                FLEX6000
                FLEX8000
                FLEX10K
                MAX5000
                MAX7000
                MAX9000
                pLSI
                ORCA1C
                ORCA2C
                CoolRunner
                pASIC1
                MACH
                XC3000
```

```
                XC4000
                XC4000E
                XC4000EX
                XC4000XL
                XC5200
                XC9500
                XC9500F
        }

# Loop through synthesis for each target technology.
        foreach technology $try_these {

# Set the target technology from the $try_these list
        set_option -technology $technology

# Run synthesis.
        project -run

# Display the Technology View showing the implementation.
        open_file -technology_view
}

# Display one RTL View showing the RTL level schematic.
# The RTL level schematic is the same for all synthesis runs since the
# design has not changed.
        open_file -rtl_view
```

5.3.2 运行多个频率要求，并存为不同的log 文件

```
# Run synthesis six times on the same design using different clock
# frequency goals. We want to see what the speed/area tradeoffs are for
# the different timing goals.

# Load an existing Project. This Project was created from an
# interactive session by saving the Project file after adding all the
# necessary files, and setting options in the Project->Implementation #Options dialog box.

project -load "design.prj"

# Create a Tcl variable, called $try_these, that will be used to
# synthesize the design with different frequencies.
        set try_these {
                20.0
                24.0
                28.0
                32.0
                36.0
                40.0
        }

# Loop through each frequency, trying each one
```

```
        foreach frequency $try_these {

# Set the frequency from the try_these list
        set_option -frequency $frequency

# Since I want to keep all Log Files, save each one. Otherwise
# the default Log File name "<project_name>.srr" is used, which is
# overridden each run. Use the name "<$frequency>.srr" obtained from the
# $try_these Tcl variable.
        project -log_file $frequency.srr

# Run synthesis.
        project -run

# Display the Log File for each synthesis run
        open_file -edit_file $frequency.srr
        }
```

5.3.3 设置控制选项及约束示例

```
# Set a number of options and use timing constraints on the design.

# Open a new Project
        project -new

# Set the target technology, part number, package, and speed grade options.
        set_option -technology XC4000E
        set_option -part XC4013E
        set_option -package PC84
        set_option -speed_grade -1

# Load the necessary VHDL files. Add the top-level design last.
        add_file -vhdl "statemach.vhd"
        add_file -vhdl "rotate.vhd"
        add_file -vhdl "memory.vhd"
        add_file -vhdl "top_level.vhd"

# Add a timing Constraint file and vendor-specific attributes.
        add_file -constraint "design.sdc"

# The top level file ("top_level.vhd") has two different designs, of
# which the last is the default entity. Try the first entity (design1)
# for this run. In VHDL, you could also specify the top level architecture
# using <entity>.<arch>
        set_option -top_module design1

# Turn on the Symbolic FSM Compiler to re-encode the state machine
# into one-hot.
        set_option -symbolic_fsm_compiler true
```

```
# Set the design frequency.
        set_option -frequency 30.0

# If you haven't switched to Xilinx M1 yet, don't forget to set the -
# xilinx_m1 option to false. The default is to write out netlists for M1.

# Save the existing Project to a file. The default synthesis Result File
# is named "<project_name>.<ext>" so, if you wanted the synthesis Result
# File to be  named something other than "design.xnf", you will change
# it with the project -result_file "<name>.xnf" command
        project -save "design.prj"

# Synthesize the existing Project
        project -run

# Open an RTL View
        open_file -rtl_view

# Open a Technology View
        open_file -technology_view

#  ------------------------------------------------
# This is a constraint file: "design.sdc". It is read by file "test3.tcl"
# with the add_file -constraint "design.sdc" command. Constraint files
# are for timing constraints and synthesis attributes.
#  ------------------------------------------------
#  Timing Constraints:
#  ------------------------------------------------
# The default design frequency goal is 30.0 MHz for four clocks. Except
# that clk_fast needs to run at 66.0 MHz. Override the 30.0 MHz default
# for clk_fast.
        define_clock {clk_fast} -freq 66.0

# The inputs are delayed by 4 ns
        define_input_delay -default 4.0

# except for the "sel" signal, which is delayed by 8 ns
        define_input_delay {sel}  8.0

# The outputs have a delay off-chip of 3.0 ns
        define_output_delay -default 3.0

# From a previous run it was noticed in Technology View that the critical
# paths are the flip-flop to flip-flop paths going to register "inst3.q[0]"
# (in the memory). Improve the paths going to inst3.q[0] by 3.0 ns.
        define_reg_input_delay  {inst3.q[0]}  -improve 3.0

#  ------------------------------------------------
#  Xilinx-specific Attribute Constraint:
```

```
# ------------------------------------------------
# Assign a location for scalar port "sel". These VHDL object names are
# case sensitive.
        define_attribute {sel} xc_loc "P139"

# Assign a pad location to all bits of a bus.
        define_attribute {b[7:0]} xc_loc "P14, P12, P11, P5, P21,
        P18, P16, P15"

# Assign a fast output type to the pad.
        define_attribute {a[5]} xc_fast 1

# Use a regular buffer, not a clock buffer for clock "clk_slow". Save the
# clock buffers for the fast clocks
        define_attribute {clk_slow} syn_noclockbuf 1

# Do not do regular buffering for "clk_slow", because its the
# slow clock and the timing can be relaxed on it.

# Set the maximum fanout to 10000.
        define_attribute {clk_slow} syn_maxfan 10000
```

5.3.4 自底向上的综合示例

```
# Bottom-up synthesis of a large design.

# The Source command reads in other Tcl scripts. Each of these scripts does
# a compile of one logic block and has its own constraint file.
        source "statemach.tcl"
        source "microproc.tcl"
        source "handshake.tcl"
        source "fifo.tcl"
        source "cherstrp.tcl"

# After synthesizing the individual logic blocks, create a Project for the
# top-level design.
        project -new

# Add the top level VHDL file.
        add_file -vhdl top_level.vhd

# Add the top level global constraint file.
        add_file -constraint top_level.sdc

# Set the top level options
        set_option -technology FLEX10K
        set_option -part EPF10K70
        set_option -speed_grade -3
        set_option -frequency 50.0
        set_option -symbolic_fsm_compiler true
```

```
# Set the output file information
        project -result_file top_level.edf
        project -log_file top_level.srr

# Save the Project to file
        project -save top_level.prj

# Run the Project
        project -run

# Open the top level RTL and Technology Views
        open_file -rtl_view
        open_file -technology_view
```

说明：以下为statemach.tcl**文件**

```
#  ------------------------------------------------
# This is file: "statemach.tcl". It is read by "bottom_up.tcl", the
# bottom up Tcl script with the command "source statemach.tcl".
# The other *.tcl scripts are similar.
#  ------------------------------------------------
# Open a new Project for "statemach"
        project -new

# Add the VHDL file for this logic block.
        add_file -vhdl statemach.vhd

# Add the constraint file for this logic block.
        add_file -constraint statemach.sdc

# Set the other options for "statemach".
        set_option -technology FLEX10K
        set_option -part EPF10K70
        set_option -speed_grade -3
        set_option -frequency 50.0
        set_option -symbolic_fsm_compiler true

# Set the Project outputs
        project -result_file statemach.edf
        project -log_file statemach.srr

# Save this Project
        project -save statemach.prj

# Run this Project
        project  -run

#  ---------------------------------------------
# This is file (statemach.sdc) is the constraint file read by
# "statemach.tcl", with the command add_file -constraint statemach.sdc.
```

```
# This constraint file is specific to this logic block " statemach "
# ------------------------------------------
# Timing Constraints:
# ------------------------------------------
        define_input_delay -default -100
        define_output_delay -default -100
        define_input_delay RESET -10
        define_reg_input_delay {q[8]} -improve 4.0

# ------------------------------------------
# Altera-Specific Attributes:
# ------------------------------------------
        define_attribute {inst1.sqrt8} altera_implement_in_eab 1
```

5.4 运行script文件

1、在UNIX环境下执行synplify -batch tcl_script_name.tcl

2、在工程窗口中选择File --> Run Tcl script

6 时间约束

约束文件是用Tcl 文件格式定义的。内容包括了通用的时间约束命令，特定的verilog 或 vhdl 属性，还包括一些由供应商提供的一些属性。

6.1 通用的时间约束

1、define_clock

语 法： define_clock [-disable] clock_name [{-freq MHz | -period ns [{-duty_ns ns | -duty_pct percent}}] [-improve ns] [-route ns]

说明：-inprove：用于提高由该时钟控制的寄存器延迟，以满足时序报告中的slack值

-route： 用于提高由该时钟控制的寄存器延迟，以满足布局布线时序报告中的时延与综合时序报告时延的差值。

define_clock 的优先级要比set_option -frequency高。

用define_clock可以定义内部时钟。

例子：
```
define_clock  clock_a  -freq 33.0
define_clock  clock_b  -perioc  100.0
define_clock  ten_mhz_internal_clock  -freq  10.0
```

2、define_clock_delay

语法：define_clock_delay [-rise | -fall] clock1 [-rise | -fall] clock2 delay

说 明：该命令既可以定义不同时钟域之间的最大延迟；定义两时钟域之间是伪路径（delay的值取-false）；定义占空比不为1的时钟。

例子：
```
define_clock  clka   -period  100.0
define_clock  clkb   -period   50.0
define_clock_delay  -rise  clka  -rise  clkb  50.0
```

define_clock_delay -fall clka -rise clkb 50.0

define_clock_delay -rise clka -rise clkb -false

define_clock clka -perioc 100

define_clock_delay -rise clka -fall clka 25

define_clock_delay -fall clka -rise clka 75

3、define_reg_input_delay

语法：define_reg_input_delay register_name [-improve ns] [-route ns]

说明：register_name**是单比特寄存器而不是寄存器总线**。

例子： define_reg_input_delay reg_a -improve 2.0

define_reg_input_delay reg_a -route 1.8

4、define_reg_output_delay

语法： define_reg_output_delay register_name [-improve ns] [-route ns]

例子： define_reg_output_delay reg_a -improve 1.0

define_reg_output_delay reg_a -route 1.8

5、define_input_delay

语法： define_input_delay {input_port_name | -default} ns [-improve ns] [-route ns]

例子： **对所有的输入端口设置**3ns**延迟**

define_input_delay -default 3.0

对输入端口input_a**设置**10ns **延迟**

define_input_delay input_a 10.0

define_input_delay input_a 0 -inprove 2.0

define_input_delay input_a 5.0 -improve 3.0

define_input_delay input_a 10.0 -route 1.8

6、define_output_delay

语法：define_output_delay {out_port_name | -default} ns [-improve ns] [-route ns]

例子： define_output_delay -default 8.0

define_output_delay output_a 10.0

define_output_delay output_a 0 -improve 2.0

define_output_delay output_a 5.0 -improve 3.0

define_output_delay output_a 10.0 -route 1.8

7、define_multicycle_path

语法： define_multicylce_path {from reg_or_input | {reg_or_output | -through reg | -from reg_or_inout -to reg_or_input} clock_cycles

说明：synplify**工具对**-from/-to**选项，只是把他当作布局布线的约束写到布局布线的文件中去。如果用**-through**开关选项，则**synplify**会把该信号设置为**syn_keep。

例子： define_multicycle_path -from register_a 2

define_multicycle_path -from register_b 2

define_multicycle_path -to register_c 2

8、define_false_path

语 法： define_false_path {-from reg_or_input | -to {reg_or_output | -through reg | -from reg_or_input -to reg_or_input}

例子：define_flase_path -from register_a

define_false_path -to register_c

9、syn_reference_clock

语法：define_attribute register syn_reference_clock clock

说明：**该命令用来定义一个参考时钟。比如一个寄存器的**en**信号的频率是它时钟频率的两倍，则可对该寄存器设置一个参考时钟为原来时钟频率的两倍。**

例子： define_clock dummy -period 40.0

define_attribute myreg[31:0] syn_reference_clock dummy

对所有使能信号为en40**的寄存器设置一个参考时钟。**

define_atrribute {find -reg -enable en40} syn_reference_clock ck2

6.2 针对黑匣子的约束命令

1、syn_tpd：**定义穿过黑匣子的组合逻辑延迟**

2、syn_tsu：**定义黑匣子的输入延迟。**

3、syn_tco：**定义黑匣子的输出延迟。**

例子：

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
component rcf16x4z port (
        ado, ad1, ad2, ad3 : in std_logic;
        dio, di1, di2, di3 : in std_logic;
        wren, wpe : in std_logic;
        tri : in std_logic;
        do0, do1, do2, do3 : out std_logic);
end component;

attribute syn_tpd1 of rcf16x4z : component is
        "ado,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1"

attribute syn_tpd2 of rcf16x4z : component is
        "tri -> do0,do1,do2,do3 = 2.0"

attribute syn_tsu1 of rcf16x4z : component is
        "ado,ad1,ad2,ad3 -> ck = 1.2"

attribute syn_tsu2 of rcf16x4z : component is
        "wren,wpe -> ck = 0.0"
```

```
-- Other coding
```

例2：

```
module ram32x4(z, d, addr, we, clk);
/* synthesis black_box
syn_tpd1="addr[3:0]->z[3:0]=8.0"
syn_tsu1="addr[3:0]->clk=2.0"
syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;

endmodule
```

6.3 书写约束文件的一些规则

1、**严格区分大小写**

2、**提供正确的目标名称**：signals，ports，instances**等**。

3、**模块名和端口名要用大括号括起来**。

4、**用圆点符隔开需要表达的层次关系**。

5、**用好恰当的前缀**：i：**表示**instance。p：**表示**port。n：**表示内部**nets。

7 怎样实现对速度的优化

第一步：**用**view log**查看**Timing**信息。如果与设计目标差距在**5％**到**10％**。可先布局布线看是否可以满足要求。再用**define_reg_input_delay -route，define_reg_output_delay -route**指导重新综合**

第二步：**在工程窗口中设置**synplify**频率设置而不是采用隐含值**0MHz

第三步：**如果设计中存在多个时钟，可在约束文件中用**define_clock**定义多个时钟，以覆盖在工程窗口中设置的工作频率**。

第四步：**用约束文件适当减少相关路径的**fan_out**数目可加快该路径工作速度**。

第五步：**如果在设计中用了有限状态机，在约束文件中打开**symbolic FSM Compiler option。

第六步：**用**HDL Analyze**查看综合结果，在相应的源代码中加入约束和属性**：

define_reg_input_delay -improve，define_reg_output_delay -improve，define_input_delay，define_output_delay**等**。**对黑匣子进行时间属性约束**：syn_tpd，syn_tsu，syn_tco**等**。

8 怎样用HDL Analyst分析和调试设计

8.1 HDL Analyst 简介

HDL Analyst**是**synplify**提供给设计者查看综合结果，提高设计的速度特性和优化面积结果的强有力的分析调试工具**。

在设计者把他的设计map到一个器件后，HDL Analyst自动生成层次化的RTL级和基本门级网表。用HDL Analyst 打开你的设计后，你就可以在你的源代码与你的逻辑图之间cross_probe了。利用这一强大的工具，你可以在源代码中选中几行代码处于高亮状态，在相关的逻辑图中你可以看到相应的逻辑图显示高亮。反之，如果你首先选中逻辑图中的一部分使它处于高亮，则相应的代码在编辑窗口中显示高亮。

8.2 HDL Analyst 的应用

8.2.1 获取信息

1，POP_UP信息

把鼠标停留在一个目标上片刻，系统会在鼠标附近显示目标的名字（instance，net，port，sheet connector等），在门级显示的情况下，打开show critical path还可以显示该路径的延迟信息。

2，状态条显示

如果你打开了View-->Status Bar信息，则把鼠标停留在一个目标上片刻，会同时在状态条上显示目标信息。

8.3 链接式选中目标

1，你可以在源代码中选中几行代码处于高亮状态，在相关的逻辑图中你可以看到相应的逻辑图显示高亮。反之，如果你首先选中逻辑图中的一部分使它处于高亮，则相应的代码在编辑窗口中显示高亮。

2，注意如果设计的代码及综合结果均处于打开状态，则只需用左键单击目标即可在另一个窗口中高亮显示相关的东西。而要求系统弹出另一种显示状态时，则需用左键双击目标。

3，注意并非所有的代码都有对应的逻辑显示，这是因为在compililation或mapping时有可能被优化掉了。

8.4 用HDL Analyst 分析关键路径

8.4.1 综合并查看延迟信息

1，在工程窗口点击RUN按钮或选择Synthesize-->Run F8 运行对设计的综合。

2，选择HDL-->Techmology view 打开综合生成的门级网表。

3，选择HDL Analyst -->show critical path 或在工具栏点击Show Critical Path 或在逻辑图显示区域按右键弹出菜单选择Show Critical Path。此时关键路径上的部件及网表节点处于高亮状态，所有的延迟信息也标在了instance上面，不过你要zoom放大才能看见。

4，把关键路径孤立出来。选择HDL Analyst-->Filter Schematic 或在工具栏选取按钮Filter Schematic等。此时系统会把关键路径上的所有元素“搜集”到一张逻辑图上，而不管这些元素原来分布在那些逻辑图或那个层次的逻辑图中。

5，再次选择 Filter Schematic 命令可以把你原始的逻辑图重新装进来。

6，灵活应用Slack Margin 命令把你想要的查看的一些关键路径而不只是一条最大延迟的路径显示到一张逻辑图中。选择Analyst -->Set Slack Margin，输入一个超出设计要求的延迟范围，比如是10ns，则所有比你定义的时钟周期大10ns的延迟路径都会显示出来。

7，正确理解关键路径上的时间延迟显示信息： out[0] (dfm7a), delay : 12.9 ns, slack: -10.5ns 该信息表示组合逻辑的延迟累积到此为12.9ns，到此已超出时间要求10.5ns。

8.4.2 怎样处理关键路径上不满足速度要求的延迟

1，关键路径的起点不是输入端就是寄存器，而结束点不是输出端就是寄存器。如果起点少就选择起点添加时间约束和时间属性。如果结束点较少就选择结束点添加时间约束和时间属性。

2，对输入端口添加约束。如果关键路径的起点是输入端口，而且该路径的slack时间是-2.4ns，则可在输入端添加约束defind_input_delay -improve 2.4。

3，如果关键路径的起点是寄存器，需对该寄存器添加时间约束和时间属性。例如：define_reg_output_delay -improve 2.4。

4，如果关键路径的终点是输出端口，则可对该该端口添加时间约束：define_output_delay 2.4

5，如果关键路径的终点是寄存器，则可添加时间约束：define_reg_input_delay 2.4。

6，对添加了约束的设计重新综合，查看延迟信息，如果时间延迟的超出部分在设计目标的5％到10％之间，则可考虑布局布线，并考虑在布局布线时添加有相应的供应商提供的约束条件。

9 怎样使用Symbolic FSM compiler

1，在工程窗口中打开Symboic FSM Compiler，则Synplify在对你的设计优化时，自动搜索你设计中的状态机，在不需要你改动源代码设计的情况下针对状态机进行优化。

2，该工具针对状态机的优化包括：对你的状态机设计重新选择编码方式加以实现；为你的状态机设计确定一个更恰当的起始状态。究竟选取什么样的编码实现取决于你对时序的要求和对面积的要求。

3，该工具还可以把状态机中多余的状态逻辑删除。例如你设计了一个只有10个状态的四位状态机，如果“1100”的状态你没有用到，则Synplify在综合时调用Symbolic FSM compiler对该状态机优化，删除与“1100”状态有关的无用逻辑。

4，该工具还可对状态机的各个状态的可达到性进行分析，对一些不能到达的状态加以删除。注意在设计状态机时一个最常见的错误便是存在永远不可到达的状态。

5，在Synplify的log file中可以查看每一个状态机的综合结果，以及每一个状态机的可达到的状态。有助于你对状态机设计实现加深认识。

6，你业可以只在局部针对状态机调用该工具。例如：

Verilog Examples:

```
reg [3:0] curstate /*synthesis state_machine */;
```

VHDL Examples:

```
signal  curstate : state_type;
attribute  syn_state_machine : boolean;
attribute  syn_state_machine of  curstate  :  signal is true;
```