第六章 门级静态时序分析与 PrimeTime 使用

一、PrimeTime 简介

PrimeTime 是 Synopsys 的一个单点的全芯片、门级静态时序分析器。它能分析大规模、同步、数字 ASICS 的时序。PrimeTime 工作在设计的门级层次，并且和 Synopsys 其它工具整合得很紧密。

基本特点和功能：

时序检查方面：

建立和保持时序的检查（Setup and hold checks）

重新覆盖和去除检查（Recovery and removal checks）

时钟脉冲宽度检查(Clock pulse width checks)

时钟门锁检查(Clock-gating checks)

设计检查方面：

没有时钟端的寄存器

没有时序约束的结束点（endpoint）

主从时钟分离（Master-slave clock separation）

有多哥时钟的寄存器

对层次敏感的时钟（Level-sensitive clocking）

组合电路的反馈环（Combinational feedback loops）

设计规则检查，包括最大电容(maximum capacitance)、最大传输时间(maximum transition)和最大扇出(maximum fanout)

PrimeTime 时序分析流程和方法：

在时序分析之前需要做的步骤：

1、 建立设计环境
   - 建立搜索路径（search path）和链接路径（link path）
   - 读入设计和库
   - 链接顶层设计
   - 建立运作条件、连线负载模型、端口负载、驱动和传输时间
2、 说明时序声明（约束）
   - 定义时钟周期、波形、不确定性(uncertainty)和滞后时间(latency)
   - 说明输入、输出端口的延时
3、 说明时序例外情况（timing exceptions）
   - 多周期路径（multicycle paths）
   - 不合法路径(false paths)
   - 说明最大和最小延时、路径分割（path segmentation）和失效弧（disabled arcs）
4、 进行分析和生成报告
   - 检查时序
   - 生成约束报告

- 生成路径时序报告

二、静态时序分析中路径延时的计算

静态时序分析工具一般将电路网表看成一个拓扑图，图中的节点(node)代表电路中的引脚(pin)。节点之间的边(edge)表示时序弧(timing arc)，有两种：

连线延时(net delay)---驱动引脚(drive pin)和扇出(fanout)之间的连接

单元延时(cell delay)---输入引脚(input pin)和输出引脚(output pin)之间的连接

延时计算就是计算每条时序弧的值，可能是单元延时也可能是连线延时。通过累计这些延时可以计算时序路径(timing delay)的上升延时(rise delay)或下降延时(fall delay)。

正函数时序弧(positive unate timing arc)：

将上升延时和上升延时相加，下降延时和下降延时相加。例如一个 AND 门单元延时和连线延时。

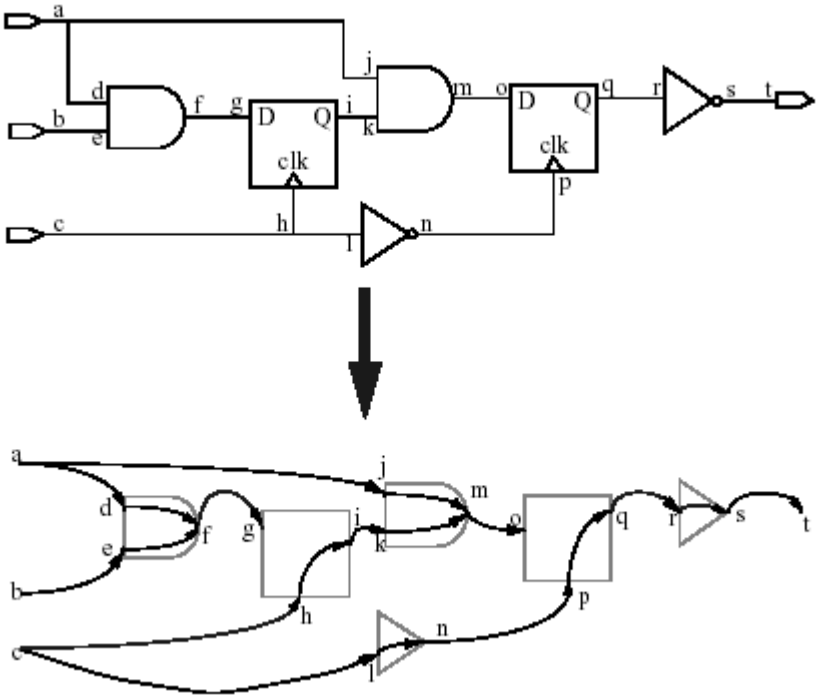
负函数时序弧(negative unate timing arc)：

将新得到的上升延时和原来的下降延时相加，而新得到的下降延时和原来的上升延时相加。例如 NAND 门。

非函数时序弧(non-unate timing arc)：

将原来的延时和新得到的最差情况延时(worst-case delay)相加。非函数时序弧出现在不能从输入量的变化预测输出端逻辑值变化的地方，例如 XOR 门。

下图展示了一个电路逻辑网络是如何转化成一张时序图的：



非线性延时模型(nonlinear delay model)：

非线性模型是供应商以查表(lookup table)形式在工艺库中提供的延时信息，它和时序分析计算有着紧密的联系。

总的延时包含了单元延时和连线延时：

Dtotal = Dcell + Dc

Dc

连线延时。它有两种计算方法，一是通过 operating_conditions 中的 tree_type 属性和 wire_load 模型；二是在标准延时方程中读入一个 SDF 文件。

Dcell

门自身的延时，典型地是取从输入引脚电压变化到 50%到输出引脚电压变化到 50%的之间的时间。

CMOS 非线性模型有两种计算 Dcell 的方法，在一个工艺库中可以混用。一是用插值法在库所提供的单元延时表里查找；二是通过查传输(propagation)表和过渡(transition)表得到传输延时和过渡延时，再计算单元延时：Dcell = Dpropagation + Dtransition 。

Dpropagation

典型衡量 Dpropagation 的方法是从输入引脚电压变化了 50%到门输出电压即将开始转变（比方说变化了 10%）之间的时间。这样，如果 Dtransition 值定义为输出电压从 10%变化到 50%之间的时间的话，它就要被加到 Dpropagation 上去。这样结果就是输入变化了 50%到输出变化了 50%之间的时间。

Dtransition

输出引脚转变状态所需要的时间，有时也指输出斜坡(ramp)时间。它是输出引脚两个参考电压之间变化的时间，可以是 20%到 80%或 10%到 50%。它是通过插值查表法得到的。

如果提供的是单元延时表，那么总延时就是：Dtotal = Dcell + Dc ；

如果提供的是传输延时表，那么总延时就是：Dtotal = Dpropagation + Dtransition + Dc 。

库单元延时时序弧的种类有：

上升传输(Rise propagation)

单元上升(Cell rise)

下降传输(Fall propagation)

单元下降(Cell fall)

上升过渡(Rise transition)

下降过渡(Fall transition)

注：每个条延时弧可以有传输延时表或单元延时表，但不能都有；同时必须有过渡延时表。

每一个延时表可以通过以下六个变量中的一个到三个查找：

input_net_transition

output_net_length

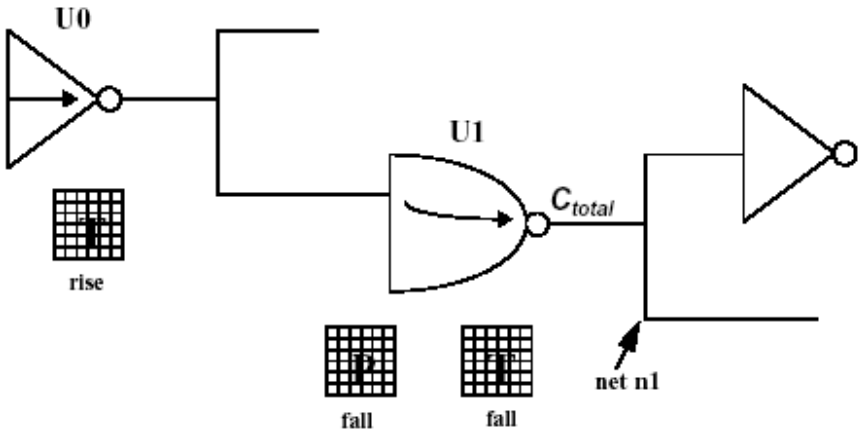
total_output_net_capacitance

related_out_total_output_net_capacitance

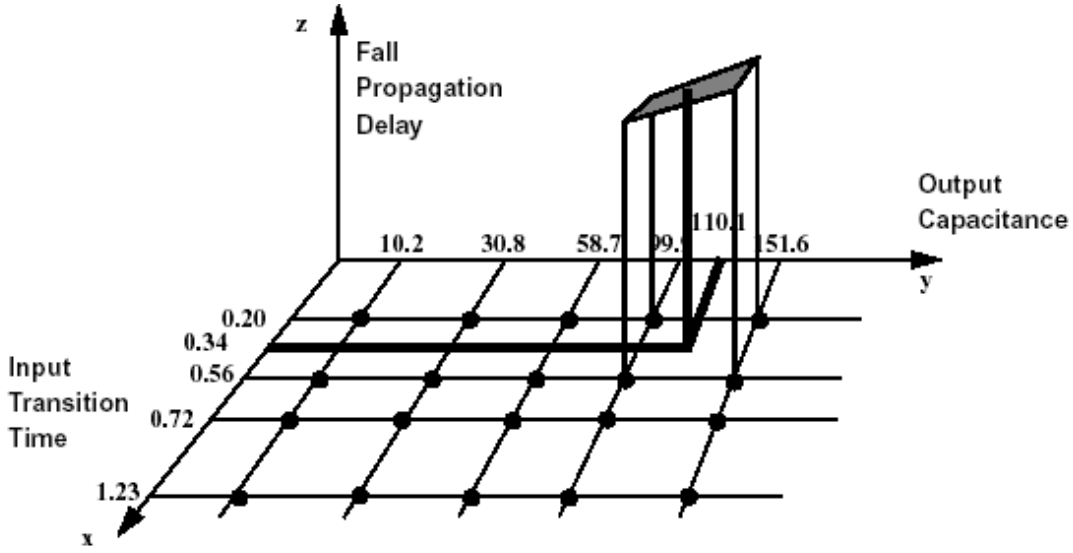
output_net_pin_cap

output_net_wire_cap

延时计算举例



看下图的下降传输(fall propagation)表：一个两维的表，由输出端总电容和输入过渡时间查找。输出端总电容由网络 n1 处的引脚电容、连线电容所决定。输入过渡时间由前面的 U0 门所决定。因为 U1 中的时序弧是负函数性质的，所以 U0 的上升过渡延时表就可以用来确定 U1 的输入过渡延时。假设 Ctotal 是 110.1，输入过渡延时是 0.34，用这两个值在下降传输延时表中查找。



图中的黑点表示表中定义的点。四个点和 Z 轴的高度值组成了供插值查找的领域，即图中的阴影部分。

下降过渡延时由下降过渡延时表得到，这个例子中它是基于输出端总电容的一维表。前面提到了输出端总电容是 110.1，通过简单的线性插值查表就可以得到延时。

然后将传输延时和下降的过渡延时相加即得到了通过 U1 单元的下降传输延时时间。

如果库中为 U1 定义的不是传输延时表，而是单元延时表，那么过渡延时时间将不计入单元延时之内。

环境缩放比例(Environmental Scaling)

当计算总延时说，时序分析器会分别考虑影响 Dtotal 的因素。每一种因素都有它自己的全局参数来反映它对总延时的影响。在通常情况下这些因素包括工艺(process)、温度(temperature)和电压(voltage)。

下面的因子可以分别应用到延时方程中去：

$$(1 + \Delta_V K_V)\ (1 + \Delta_T K_T)\ (1 + \Delta_p K_p)$$

△v：库中定义的电压变化的值

Kv：电压对总延时影响的因子

△t：库中定义的温度变化的值

Kt：温度对总延时影响的因子

△p：库中定义的工艺变化的值

Kp：工艺对总延时影响的因子

三、PrimeTime 的基本概念

1、 定义设计环境

在对设计作时序分析之前，必须要定义好设计环境以使得在那些情况下满足
限制条件。

通过以下这些信息来说明设计环境：

时钟：时钟波形和时钟信号的性质；

输入、输出延迟：信号到每个输入端口的时间从每个输出端口离开所需的时间。这些时间是用一个时钟周期的相对量表示的；

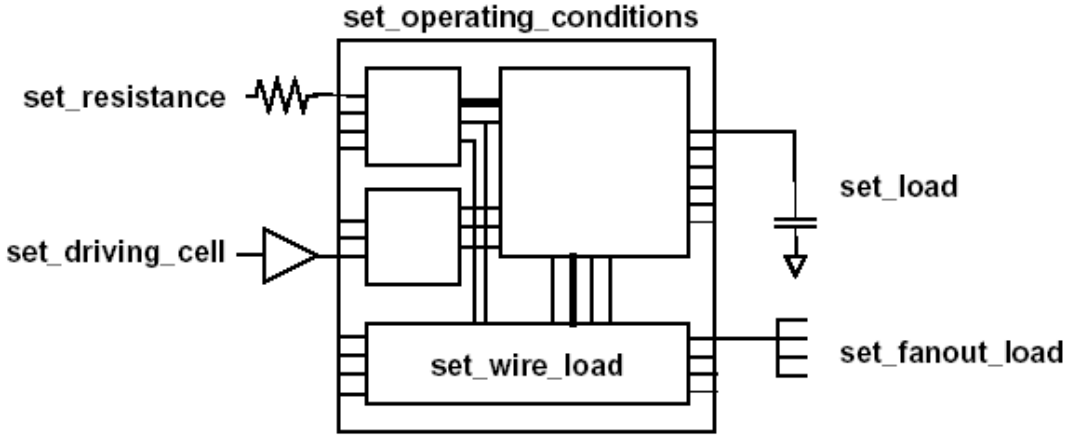
输入端口的外部驱动：每一个输入端口的驱动单元或驱动电容，还可以用一个确定的过渡时间来表示；

电容负载：输入或输出端口的外部电容；

运作条件：环境特性（工艺、温度和电压）；

连线负载电容：用来预测布局布线后每一条连线的电容和电阻。

下图展示了用来定义设计环境的命令：



2、 时序声明

通常当前设计只是一个更大电路的一部分。时序声明提供了时钟和输入、输出延时的信息。在将设计建立起来之后，可以进行时序声明。

为了进行时序声明，包括以下一些内容：

说明时钟信息

描述一个时钟网络

说明时钟门锁（Clock-Gating）的建立和保持时间（Setup and Hold Checks）

建立内部生成的时钟

说明输入延时

说明时钟端的输入延时

说明输出延时

3、 时序例外（Timing Exceptions）

PrimeTime 缺省地认为所有的电路都是单时钟周期的。这意味着电路在一个时钟周期之内将数据从一条路径的开始端传递到结束端。

在某些情况下，电路不是工作在这样的方式下。对具体的一条路径来说不适用单始终周期时序，所以必须对这些缺省的时序假设作例外说明。否则，时序分析将不能反映真实电路的工作情况。

主要有以下一些内容：

单时钟周期（缺省）路径延时限制
设置失败（False）路径
设置最大和最小路径延时
设置多时钟周期路径
路径说明方法
有效地说明例外情况
例外情况的优先级
报告例外情况
忽略例外情况
去除例外声明

4、 报告的生成

在定义了时序声明和例外情况之后，可以生成时序分析报告，有助于定位设

计中的违规之处。在进行时序分析的时候，PrimeTime 会跟踪电路中所有的路径然后根据电路说明、库、声明和例外情况计算设计的延时。

有以下一些内容：

检查设计约束
报告时序检测的覆盖率
生成路径时序报告
去除有寄存器的路径上的时钟扭斜(Skew)
生成瓶颈报告
进行快速时序升级（Fast Timing Updates）
生成约束报告
生成设计信息报告
生成连线负载报告
生成时序例外情况报告
报告最大扭斜检查（Maximum Skew Checks）
报告不变的时序检查（No-Change Timing Checks）
报告失效的时序弧（Disabled Timing Arcs）
显示情形分析设置
观察扇入逻辑
观察扇出逻辑
显示层次参考（Hierarchical References）
报告单元参考（Cell References）

生成总线报告
生成反标延时和检查报告（Annotated Delay and Check Reports）
生成模式分析报告（Mode Analysis Reports）
生成库的报告
生成延时计算报告
以路径（Paths）来生成定制报告
禁止和恢复时钟门锁、去除检查时钟门锁
以弧（Arcs）来生成定制报告

5、 高级分析

用 PrimeTime 可以进行各种类型的高级分析。
内容有：
单运作条件分析（Single Operating Condition Analysis）
最小和最大分析
用片上变量（On-Chip Variation）分析设计
分析模式摘要
情形分析（Case Analysis）
模式分析（Mode Analysis）
检测失败路径
层次敏感的、基于锁存器的设计
分析异步逻辑的设计
分析三态总线的设计

6、 读写 SDF

对于起初的静态时序分析，估计网络的延时信息是基于一个连线负载模式。实际上延时是与设计中单元和网络的布局布线有关的。

一个布局器或一个布线器提供更详细和更精确的延时信息，可以用来提供给 PrimeTime 作更精确的分析。这个过程被称作反标（back-annotation）。反标信息经常是以标准延时格式 Standard Delay Format(SDF)提供的。

包括以下信息：
读入一个 SDF 文件
报告延时反标信息
用 SDF 标注条件延时（Conditional Delays）
写一个 SDF 文件
用 PrimeTime 写 SDF
去除标注延时和检查
用命令行设置标注
生成布局和布线的时序约束
为整个设计提供约束覆盖

可以用以下一些方法来读取 SDF 的反标延时信息：
从一个 SDF 文件里读取延时和时序检查

用命令行、而不用 SDF 标注延时、时序检查和翻转时间

7、 反标寄生信息

PrimeTime 为延时计算提供了增强的精确度。

PrimeTime 为集中参数电容、集中参数电阻、精简 pi 模型和详细 RC 网络提供了寄生反标信息。

有以下一些内容：

寄生标注支持的文件格式

标注集中寄生效应

标准精简寄生效应和详细寄生效应

精简 pi 模型和详细 RC 网络比集中参数电容和电阻精确得多，但是需要建立环境变量并且会占用更多的 CPU 时间和内存。为了节约 CPU 时间推荐使用一个 SDF 文件，因为 PrimeTime 不必去计算延时。

8、 编辑网表

PrimeTime 为编辑网表提供了命令。编辑网表是为了在不违背逻辑综合的前提下满足时序要求。

9、 相关特性鉴定（Context Characterization）

相关特性鉴定是从一个子设计的环境和它的上级设计来提取它的时序特性。鉴定相关特性有两个主要应用：

作为 DC 的脚本：鉴定所得的相关特性在 DC 综合或逻辑优化时设置时序约束。

在 PrimeTime 内部：在研究芯片层次的时序约束时鉴定得到的相关特性可以用来作层次化的时序分析。

10、 生成快速时序模型

快速时序模型是临时的时序模型，可以用来快速描述时序信息而不需要用标
志建模语言(Stamp modeling language)来写一个模型。

在设计周期早期用快速时序模型来大致描述没有定义的模块的初始时序。快速时序模型最终要被标志模型(Stamp models)或门级网表代替，因为它们含有更精确的时序信息。

四、PrimeTime 使用

开始

先建立目录并将 PrimeTime 本身所带的一个例子拷到新建的目录下，在下面的内容中将要用到这个例子。

mkdir primetime

cd primetime

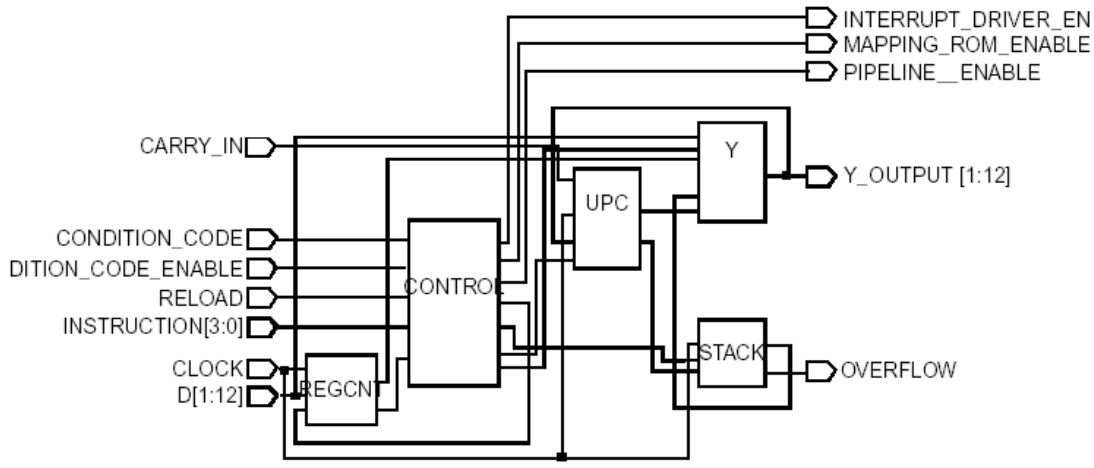
cp –r $SYNOPSYS/doc/pt/tutorial .

cd tutorial

确认目录中有以下这些文件：

AM2910.db	The design .db for the top-level of the design
CONTROL.db	The design .db for the CONTROL block
REGCNT.db	The design .db for the REGCNT block
UPC.db	The design .db for the UPC block
Y.data	The Stamp data file for the Y block
Y.mod	The Stamp model file for the Y block
Y_lib.db	The library .db for the Y block
STACK_lib.db	The library .db for the STACK block
pt_lib.db	The technology library .db
stack.qtm.pt	The quick timing model script for the stack block
optimize.dcsh	The dc_shell optimization script
timing.dcsh	An example DC shell timing script for translation
tutorial.pt	The complete PrimeTime tutorial script for your reference.

例子是一个 AM2910 微处理器，如图所示模块图。



运行 PrimeTime：

pt_shell

定义搜索路径和链接路径：

pt_shell>set search_path “.”

Pt_shell>set link_path “ pt_lib.db STACK_lib.db Y_lib.db”*

* *pt_lib.db STACK_lib.db Y_lib.db*

读入设计：

PrimeTime 支持以下设计格式：

. Synopsys database files (.db) (Use the read_db command)

. Verilog netlist files (Use the read_verilog command)

. Electronic Design Interchange Format (EDIF) netlist files (Use the read_edif command.)

. VHDL netlist files (Use the read_vhdl command.)

读入 AM2910 的顶层设计文件：

pt_shell> read_db AM2910.db

Loading db file '/u/joe/primetime/tutorial/AM2910.db'

1

链接设计：

pt_shell> link_design AM2910

Loading db file '/u/joe/primetime/tutorial/pt_lib.db'

Loading db file '/u/joe/primetime/tutorial/STACK_lib.db'

Loading db file '/u/joe/primetime/tutorial/Y_lib.db'

Linking design AM2010 ...

Loading db file '/u/joe/primetime/tutorial/STACK.db'

...

Designs used to link AM2910:

CONTROL, REGCNT, STACK, UPC, Y

Libraries used to link AM2910:

STACK_lib, Y_lib, pt_lib

Design 'AM2910' was successfully linked

1

显示当前已载入的设计：

pt_shell>list_designs

得到当前载入单元的信息：

pt_shell>report_cell

编译一个标记模型（Stamp Model）：

标记模型是一个诸如像 DSP 或 RAMS 那样复杂模块的静态时序模型。

标记模型与.lib 模型共存，而不能代替它们。

- 建立标记模型是用在晶体管层次的设计上，在这个层次上没有门级网表。
- 标记模型语言是一种源代码语言，被编译成 Synopsys 的.db 文件格式，可以被 PrimeTime 或 Design Compiler 使用。
- 标记模型包含引脚到引脚的时序弧、建立和保持时间数据、模式信息、引脚的电容和驱动能力等等。标记模型还能保存属性（面积等等）。
- 三态输出、锁存器和内部生成的时钟都可以被建模。

一个标记模型包括两种源代码文件格式：

- .mod 文件
  仅包含引脚到引脚的弧的描述（没有延时数据）。
- .data 文件
  包含.mod 文件中每条弧的延时数据。

标记模型可以有多个.data 文件来描述不同运作条件下的时序。

两种文件格式都需要编译成一个.db 模型。

编译 AM2910 中 Y 模块的标记模型（标记源代码文件是 Y.mod 和 Y.data）：

```
pt_shell> compile_stamp_model -model_file Y.mod \
                -data_file Y.data -output Y
Wrote model library core to ‘./Y_lib.db’
Wrote model to ‘./Y.db’
```

PrimeTime 生成两个.db 文件：

Y_lib.db：一个库文件，包含一个单元(cell)。这个单元叫做核(core)。

Y.db：一个设计文件，引用 Y_lib.db 中的单元核。

编译一个快速时序模型(Quick Timing Model)：

可以为设计中还没有完成的模块建立一个快速时序模型，以使得完整的时序分析能够进行。通常的情形是：

- 模块的 HDL 代码还没有完成时
- 为了划分设计，在评估阶段为实际设计进行时序预测、约束估计时
- 模块的标记模型还没有完成时

一个快速时序模型是一组 PrimeTime 命令，而不是一种语言。为了方便和文档化可以将它们写在一个脚本文件中，然后保存为.db 的格式。在 PrimeTime 和 Design Compile 中快速时序模型很有用处。

还可以将快速时序模型保存为标记模型，这是开始一个复杂标记模型的一种便利的方法。

例子中 STACK 模块的快速时序模型脚本文件是 stack.qtm.pt，建立这个模型：

```
pt_shell> source -echo stack.qtm.pt
...
pt_shell> report_qtm_model;
...
pt_shell> save_qtm_model -output STACK -format db
Wrote model library core to './STACK_lib.db'
Wrote model to './STACK.db'
1
```

进行时序分析

配置运作环境

读入并链接 AM2910 设计：

```
pt_shell> set search_path "."
pt_shell> set link_path "* pt_lib.db STACK_lib.db Y_lib.db"
pt_shell> read_db AM2910.db
pt_shell> link_design AM2910
```

链接了AM2910会导致其它已经链接的设计变为不链接的状态。在内存里只允许有一个链接的设计。当一个设计不链接，所有时序信息将被去除，并会出现警告，这和Design Compiler不同。如果需要保存所标注的信息，可以在链接一个新的设计之前用write_script命令。如果以后重新链接这个设计，只要运行这个脚本就可以了。

建立运作条件和连线负载模型：

PrimeTime在生成建立时序报告(setup timing reports)时使用最大(Maximum)运作条件和连线负载模型；在生成保持时序报告(hold timing reports)时使用最小(Minimum)运作条件和连线负载模型。

```
pt_shell> set_operating_conditions -library pt_lib -min BCCOM -max
WCCOM
pt_shell> set_wire_load_mode top
pt_shell> set_wire_load_model -library pt_lib -name 05x05 -min
pt_shell> set_wire_load_model -library pt_lib -name 20x20 –max
```

如果运作条件在两个不同的库中，用set_min_library命令来在最大库和最小库中建立联系。

得到一张库的列表：

```
pt_shell> list_libraries
Library Registry:
STACK_lib /home/gray/primetime/tutorial/
STACK_lib.db:STACK_lib
Y_lib /home/gray/primetime/tutorial/Y_lib.db:Y_lib
* pt_lib /home/gray/primetime/tutorial/
pt_lib.db:pt_lib
```

得到一个库的详细信息：

```
pt_shell>report_lib  pt_lib
```

基本声明：

```
pt_shell> create_clock -period 30 [get_ports CLOCK]
pt_shell> set clock [get_clock CLOCK]
pt_shell> set_clock_uncertainty 0.5 $clock
pt_shell> set_clock_latency -min 3.5 $clock
pt_shell> set_clock_latency -max 5.5 $clock
pt_shell> set_clock_transition -min 0.25 $clock
pt_shell> set_clock_transition -max 0.3 $clock
```

时钟门锁检查(Clock-Gating Checks)：

pt_shell> set_clock_gating_check -setup 0.5 -hold 0.1 $clock

pt_shell> set_min_pulse_width 2.0 $clock

如果设计被反标过，PrimeTime用SDF的建立和保持时间值，以及时钟脉冲宽度说明。

得到一个时序摘要：

pt_shell>report_design

…

pt_shell>report_reference

…

检查时序声明和设计的结构：

在进行时序分析之前运行 check_timing 命令是关键。这个命令能够检查到所有可能的时序问题。

在这个例子中将会出现警告，原因是存在没有约束条件的端口。

运行时序分析

设置端口延时并检查时序：

```
pt_shell> set_input_delay 0.0 [all_inputs] -clock $clock
pt_shell> set_output_delay 2.0 [get_port INTERRUPT_DRIVER_ENABLE] -clock $clock
pt_shell> set_output_delay 1.25 [get_port MAPPING_ROM_ENABLE] -clock $clock
pt_shell> set_output_delay 0.5 [get_port OVERFLOW] -clock $clock
pt_shell> set_output_delay 1.0 [get_port PIPELINE_ENABLE] -clock $clock
pt_shell> set_output_delay 1.0 [get_port Y_OUTPUT] -clock $clock
pt_shell> set_driving_cell -lib_cell IV -library pt_lib [all_inputs]
pt_shell> set_capacitance 0.5 [all_outputs]
pt_shell> check_timing
```

保存设置：

将所设置的时序信息保存为脚本文件可以确保在接下去的运行中保留一个时序环境的复本。使用write_script命令，这个命令将以下这些信息保存到一个命令文件中：

Clocks	Names, waveforms, latency, and uncertainty
Exceptions	False and multicycle paths, minimum and maximum delays, and path groups
Delays	Input and output delays, all delay annotations, and timing checks
Net and port attributes	Capacitance, resistance, and fanout
Design environment	Wire load model, operation condition, drive, driving cell, and transition

Design rules　　Minimum and maximum capacitance, minimum and maximum fanout, and minimum and maximum transition

write_script命令可以将脚本写成Design Compiler格式(dcsh or dctcl mode)或者PrimeTime格式(ptsh)。

不能用PrimeTime写一个被标注设计的.db文件，因为PrimeTime只能写时序模型的.db文件。以脚本为主要方式是为了和Design Compiler传递数据。

```
pt_shell> write_script -format dctcl -output AM2910.tcl
pt_shell> write_script -format dcsh -output AM2910.dcsh
pt_shell> write_script -format ptsh -output AM2910.pt
```

运行基本的分析：

1．得到AM2910的约束报告：

pt_shell>report_constraint

2. 检查报告中的时序违规(timing violations)和约束违规（constraints violations）

3．到更多关于违规的信息：

pt_shell> report_constraint -all_violators

4．检查这份报告：

这里有多少违规的结束点(endpoints)？

报告基于路径的时序信息：

pt_shell>report_timing

设置时序例外情况：

因为PrimeTime直到进行完整的时序升级(timing update)之前才检查时序例外情况的正确性，所以要运行report_exceptions以确定它们的正确性。

声明AM2910的时序例外情况。设置一条两个时钟周期的路径，其中建立时间为2，保持时间为1：

```
pt_shell> set_false_path -from U3/OUTPUT_reg[*]/CP \
-to U2/OUTPUT_reg[*]/D
pt_shell> set_multicycle_path -setup 2 -from \
INSTRUCTION[*] -to U2/OUTPUT_reg[*]
pt_shell> set_multicycle_path -hold 1 -from \
INSTRUCTION[*] -to U2/OUTPUT_reg[*]
pt_shell> update_timing
pt_shell> report_exceptions
pt_shell> report_exceptions -ignored
```

评估时序例外情况结果：

1．得到另外一个约束报告并评估违规情况：

pt_shell>report_constraint –all_violators

2．检查这份约束报告

3．确信所设置的例外情况是否能够使得设计中的违规显现比以前更少。设计中最

差余量(the worst slack)是什么?

pt_shell>report_timing

4. 检查这份详细的时序报告。
5. 看其它的违规路径。从这份约束报告中选择一个结束点，并键入：

   pt_shell> report_timing -to endpoint
6. 检查这份报告。

提取一个子设计的边界时序特性信息：

一个子设计的相关时序特性信息的提取是基于它相关的上级设计环境。这些信息有两个主要用途：

一个主要用途是PrimeTime将这些相关特性信息提供给Design Compiler作为约束信息。这是DC中从芯片级分析到模块级优化主要的纽带，与只是作分析的单点工具相比提供了更佳的整合性。

提取了相关特性信息之后，命令PrimeTime写一个包含子设计或模块时序信息的脚本。

在提取时序特性的同时，要注意：

1. 特性信息提取不能够预计子设计的时序状况；
2. 特性信息提取没有最大或最小这两种工作模式。所以在做这步工作之前要设置单一的、正确的运作条件。

对于PrimeTime而言，相关时序信息允许作层次化的时序分析，并观察芯片层次的时序约束；对于Design Compiler而言，相关时序信息允许在综合或逻辑优化时设置时序约束。

在Design Compiler中设置综合或优化约束的步骤是：

1. 在PrimeTime中读入顶层设计
2. 确认需要优化的子设计
3. 提取每一个子设计的特性信息
4. 为每一个子设计生成一个Design Compiler的脚本
5. 将这些子设计读到Design Compiler中
6. 将步骤4中生成的脚本引入进来
7. 进行模块级的优化

仅将需要优化的子设计读入到DC中去，这样DC运行起来可以效率高一些。优化完这些模块之后，再将它们读入到PrimeTime中去作新一轮的时序分析。

在这个例子中，从时序报告中可以看出模块U3(REGCNT)和U2(UPC)可以进一步优化，也许能消除一些违规情况。

因为要纠正建立时序违规，所以要配置最差情况的运行条件。

```
pt_shell> set_operating_conditions -library pt_lib WCCOM
pt_shell> characterize_context {U2 U3}
pt_shell> write_context U2 -output UPC.char.dcsh \
-format dcsh
pt_shell> write_context U3 -output REGCNT.char.dcsh \
-format dcsh
pt_shell> write_script -format ptsh -output AM2910.new.pt
```

```
% dc_shell
```

dc_shell> include optimize.dcsh
...
dc_shell> quit

pt_shell> read_db {REGCNT.opt.db UPC.opt.db}
pt_shell> current_design AM2910
pt_shell> swap_cell U3 {REGCNT.opt.db:REGCNT}
pt_shell> swap_cell U2 {UPC.opt.db:UPC}
pt_shell> source AM2910.new.pt
pt_shell> check_timing
pt_shell> report_constraint -all_violators
pt_shell> report_constraint -all_violators –verbose
看新生成的报告，违规情况是不是比原来少了？

高级分析

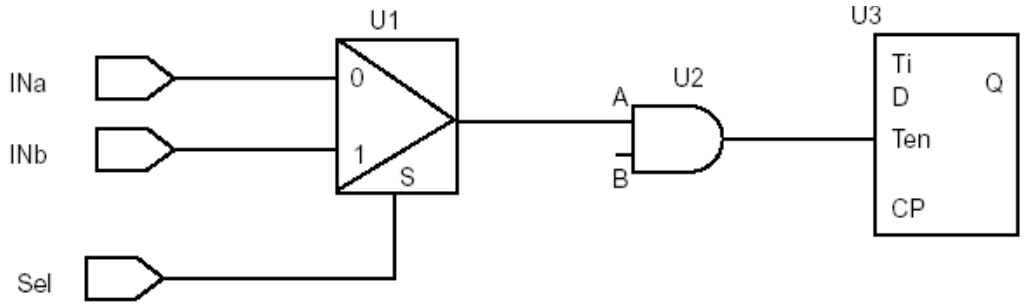
情形分析(Case Analysis)：

PrimeTime允许将设计中的端口设置成逻辑1或逻辑0，并使其像实际中那样生效，恰当地使时序弧有效或无效。这叫做情形分析(case analysis)或常量传播(constant propagation)。

情形分析能沿着电路正向地使所指定的逻辑常量生效，但是逆向不行。PrimeTime可以这样做是因为它知道门的逻辑功能。PrimeTime不能使逻辑常量通过RAMs或其他黑箱单元传播。黑箱单元是没有定义功能的单元。

可以使用标记时序模型有条件地定义被情形分析影响到的时序弧。使用情形分析，可以在不同的条件下进行时序分析，例如，测试模式的开或关。

PrimeTime自动使一直高或一直低的信号生效。如图所示使用情形分析时的常量传播。



如果将Sel端口设置成逻辑0，PrimeTime只跟踪INa到U3的路径；

如果将Sel端口设置成逻辑1，PrimeTime只跟踪INb到U3的路径；

如果将U2的B引脚设置成逻辑0，PrimeTime将不跟踪到U3/Ten的路径。

作为情形分析的范例，完成以下一些步骤：

1．在设计中的一个端口上设置一个情形分析逻辑常量，观察时序弧受到的影响

pt_shell> set_case_analysis 0 [get_ports CONDITION_CODE]

pt_shell> report_case_analysis

pt_shell> report_disable_timing

report_disable_timing命令显示所有因为情形分析而无效的时序弧。

2．看时序的改变

pt_shell> report_constraint

pt_shell> report_timing

3．检查报告

在这种情况下，将CONDITION_CODE端口设置成0改变了时序，所以关键路径也不一样了。

4．将CONDITION_CODE设置成1，观察结果

5．去除刚才所设置的逻辑常量

pt_shell> remove_case_analysis [get_ports CONDITION_CODE]

模式分析(Mode Analysis)：

一些复杂的设计可能会有好多种功能模式，在每种模式种时序路径和特性完全不同。

在PrimeTime中可以定义和说明这些模式的时序，然后再为每种模式分别进行分析。这样做可以去除许多不合法的时序违规现象，因为那些路径被设置成是无效的。例如，一个RAM的写地址和读地址路径是不同的。一个时序报告可能会显示一条RAM的写地址路径，但是这条路径只有在RAM工作在读模式时才有效。

有两种方法定义模式：

. 在标志模型中将模式和时序弧联系起来

. 为一条特殊的路径定义一种模式

在定义了模式之后，可以用一部分或所有定义的模式来进行时序分析。

1. AM2910的Y模块有模式功能，因为在它的标记模型中已经定义了。在Y.mod文件中查看已定义的模式：

   pt_shell>report_mode

2. 看设置了模式之后时序的改变：

pt_shell> set_case_analysis 0 [get_pins U4/OPERATION[*]]

pt_shell> set_mode data U4/core

pt_shell> report_mode

*pt_shell> report_timing -to Y_OUTPUT**

pt_shell> set_mode stack U4/core

pt_shell> report_mode

*pt_shell> report_timing -to Y_OUTPUT**

3. 将所定义的模式复位：

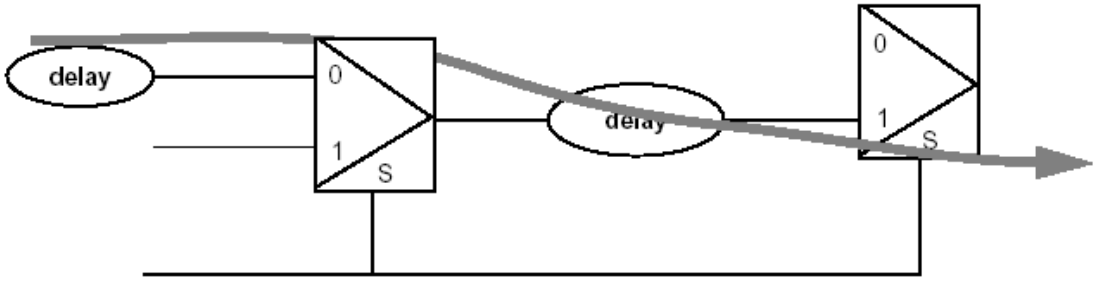
   pt_shell>reset_mode

报告合法路径(True Paths)：

PrimeTime能够自动探测到设计中存在的一些不合法路径。这些路径可能是功能不合法

路径或是延时不合法路径。

下图所示一条功能不合法路径，因为它永远也不敏感(sensitize)。



PrimeTime还可以用自动生成测试模式automatic test-pattern generation(ATPG)方法在需要测试的时序路径上生成测试向量来进行分析。用户不用自己去说明这些向量，PrimeTime会自动生成并使其生效。

如果PrimeTime能够生成一个向量，它会认为这条路径是合法的，否则是不合法的。

PrimeTime不能在包含有黑箱单元的部分使用合法路径分析，因为它无法推算

ATPG向量经过不知道功能的黑箱单元之后将是什么样的输出。

report-timing 命令有三个选项可以用来作合法路径报告：

-justify

对于所需分析的路径，报告每一条是合法还是不合法。如果合法，PrimeTime显示一条可以使其敏感的输入向量。将这个选项用在违规路径上查看潜在不合法违规。加上-nworst和-max_paths选项检查多条路径。

-true

启用一种搜索算法寻找最长的合法路径。使用这个选项在某些设计中会使CPU的运行时间延长。

-false

只报告不合法的路径。

仅在生成报告时使用合法路径分析。它不是PrimeTime的一种时序模型。

在例子中用合法路径报告来验证路径：

pt_shell> report_timing -true

pt_shell> report_timing -justify -to MAPPING_ROM_ENABLE

pt_shell> report_timing -false -max_paths 5

提取一个时序模型：

时序模型提取是从一个门级网表生成一个.db文件。PrimeTime和DC都能够使用这种提取得到的.db文件。

供应商(Vendors)用提取的方法提供时序模型。设计者可以用提取的方式生成一个完全与其他模块不相关的时序模型（可以被多次引用）。

提取是自动地从网表中提取时序模型并把信息保存为两个.db文件，类似于编译标记模型时的输出。

用这种方法提取一个时序模型：

1. 读入要提取的网表
2. 定义时钟
3. 反标延时和电容（如果可行的话）
4. 设置连线负载模型（如果可行的话）
5. 去除内部的例外情况（这些不保存到模型中）
6. 进行时序检查并改正任何错误
7. 设置提取时的环境变量
8. 在所有需要的运作条件下提取模型
9. 交换(Swap)提取的时序模型

现在UPC设计已经优化了，提取它的一个时序模型，用到芯片级的时序分析中去。

pt_shell> link_design UPC

pt_shell> create_clock -period 30 [get_ports CLOCK]

pt_shell> set_wire_load_model -name 20x20 -library pt_lib UPC

pt_shell> check_timing; # Unconstrained outputs are OK

pt_shell> set extract_model_tolerance 0.05

pt_shell> set extract_model_transition_limit 5.0

pt_shell> set extract_model_capacitance_limit 64

pt_shell> set extract_model_min_resolution 0.1

pt_shell> set extract_model_min_delay_threshold 0.5

pt_shell> set extract_model_conservative true

pt_shell> extract_model -operating_conditions {WCCOM NOM BCCOM} -output UPC.ext

pt_shell> set link_path "$link_path UPC.extr_lib.db"

pt_shell> read_db UPC.extr.db

pt_shell> link_design AM2910

pt_shell> swap_cell U2 {UPC.extr.db:UPC}

pt_shell> source AM2910.new.pt

pt_shell> report_cell; # Note UPC is now marked as a model

pt_shell> report_constraint -all

检查结果。

反标标准延时格式文件(SDF)：

PrimeTime支持以下版本的SDF文件：

读：1.0, 2.0, 2.1；

写：1.0, 2.1。

当PrimeTime用SDF进行延时标注时，它假定与负载相关的延时部分被包含到单元延时中。如果SDF在连线延时中包含了负载相关延时，在read_sdf命令中用-load_delay net选项。

写一个AM2910的最大和最小延时SDF文件：

pt_shell> set_operating_conditions -library pt_lib -min BCCOM -max WCCOM

pt_shell> write_sdf -version 2.1 AM2910.sdf

将这个SDF文件读回PrimeTime中：

pt_shell> read_sdf -min_max AM2910.sdf

pt_shell> report_timing

检查报告。注意每一个反标延时都被标上了*号。

读寄生参数文件：

PrimeTime支持以下几种寄生参数文件：

Reduced Standard Parasitic Format (RSPF)

Detailed Standard Parasitic Format (DSPF)

Standard Parasitic Exchange Format (SPEF)

命令格式为：

pt_shell> read_parasitics filename

PrimeTime会自动识别文件的格式。

退出PrimeTime：

pt_shell> quit