赛 Tcl在Vivado中的使用

2014-07-28 09:52 发表 系统分类：EDA 自定义分类：vivado笔记
标签：Xilinx Vivado

Vivado是Xilinx最新的FPGA设计工具，支持7系列以后的FPGA及Zynq 7000的开发。与之前的ISE设计套件相比，Vivado可以说是全新设计的。无论从界面、设置、算法，还是从对使用者思路的要求，都是全新的。看了大家很多的博文，基本上都是用GUI创建工程，那我就简单介绍一下Vivado的脚本使用。

在ISE设计套件中，支持多种脚本： 可以用xperl来运行perl脚本，可以用xtclsh来运行Tcl脚本，还可以用windows批处理脚本来运行设计流程。

ISE集成的Tcl脚本解释器为8.4版本。同时，ISE GUI中的Tcl console功能不够强大，部分组件使用的脚本也与Tcl有不同，导致Tcl脚本在ISE上并不十分流行。

在Vivado上，Tcl已经成为唯一支持的脚本。并且，所有操作都有对应的Tcl脚本可以执行。所以，掌握Tcl脚本语言对掌握Vivado的使用有重要帮助。

Vivado上集成的Tcl脚本解释器为8.5版本，也是目前比较流行的Tcl版本。Vivado的核心就是一个脚本解释器，GUI界面只是将各种脚本命令封装为图形化界面而已。

下面以Windows为平台，用脚本的思路，运行一下Vivado：
首先需要设置环境变量，在path环境变量中添加Vivado的路径，路径设置到bin文件夹，例如 C:\Xilinx\Vivado\2014.1\bin
在Windows界面下，“开始”->“运行”，输入cmd，打开windows命令行终端。这个时候 有三个选择：

1. 输入“vivado”，启动Vivado GUI界面，和点击桌面上的图标启动Vivado没什么区别；事实上，直接点击桌面图标，就是调用windows batch命令启动vivado
2. 输入“vivado -mode batch -source file.tcl”，从脚本批处理的形式启动Vivado,运行后直接执行file.tcl文件
3. 输入“vivado -mode tcl”，启动Tcl交互式命令行。

使用第三种方法。启动后显示Vivado的版本，这里使用2014.1

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\willl>vivado -mode tcl

****** Vivado v2014.1 (64-bit)
  **** SW Build 851449 on Sun Mar  2 19:43:12 MST 2014
  **** IP Build 851351 on Sun Mar  2 11:33:03 MST 2014
    ** Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.

Vivado%
```

输入命令“info tclversion”可以查看到Vivado使用的Tcl的版本 8.5

```
Vivado% info tclversion
8.5
Vivado% _
```

Tcl是一种很容易用户自己定义命令的脚本语言，Xilinx在此基础上增加了大量Vivado的命令。对于Vivado自定义的非标准的Tcl命令，输入该命令后，继续输入空格+“-help”，可以查到该命令的详细解释。

Vivado GUI中的Tcl console和CMD启动的交互命令行功能基本相同，不同在于Vivado 在切换路径时可以利用CMD的自动补缺功能更方便的切换路径。

Vivado有两种设计流程：project模式和non-project模式。

如果已经有设计工程了，可以使用Tcl脚本继续进行project的流程，例如：

```
open_project TEST.xpr     #打开已有的工程文件TEST.xpr
launch_runs synth_1       #运行综合 synth_1
wait_on_run synth_1       #等待综合结束
launch_runs impl_1 -to_step write_bitstream     #运行实现impl_1，并生成bit文件
wait_on_run impl_1        #等待实现结束
```

如果使用non-project模式，则脚本会复杂一些，下面提供一个模板。

注：英文注释是参考Xilinx相关文档，中文注释是为了方便阅读，由于Vivado原生不支持中文，所以为了避免不必要的错误，建议使用时去除中文

```
##########################################################################
##########################################################################
##########################################################################
# STEP#1: define the output directory area.
# 定义工程文件的存放路径
set outputDir ./PRJ
# file mkdir $outputDir
#
# STEP#2: setup design sources and constraints
#
# VHDL
#
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ] #指定需要添加的VHDL库文件，
read_vhdl ./Sources/hdl/bft.vhdl              #指定需要添加的VHDL文件
# ##############################
# Verilog HDL
#
read_verilog  [ glob ./SRC/*.v ]                #指定需要添加的Verilog文件，glob是扫描某个路
# ##############################
# XDC
#
read_xdc [ glob ./CONSTRS/*.xdc ]               #指定需要添加的xdc文件，glob是扫描某个路径下
# ##############################
# EDIF and NGC
#
read_edif ../test.edif                          #指定需要添加的网表文件
# ##############################
# IP XCI
#
read_ip ./CORE/MMCM/MMCM.xci                    #指定需要添加的xci IP文件
# ##############################
# STEP#3: run synthesis, write design checkpoint, report timing,
# and utilization estimates
# 运行综合 ，同时设定相关综合参数
synth_design -top SCRIPT_TEST       \
             -part xc7z100ffg900-2  \
             -fanout_limit 1000     \
             -shreg_min_size 3      \
             -flatten_hierarchy full
write_checkpoint -force $outputDir/post_synth.dcp #存档
report_timing_summary -file $outputDir/post_synth_timing_summary.rpt    #生成时序报告
report_utilization -file $outputDir/post_synth_util.rpt #生成资源使用报告
#
#
# STEP#4: run logic optimization, placement and physical logic optimization,
# write design checkpoint, report utilization and timing estimates
#
opt_design         #优化设计
place_design       #布局
report_clock_utilization -file $outputDir/clock_util.rpt    #生成资源使用报告
write_checkpoint -force $outputDir/post_place.dcp       #存档
report_timing_summary -file $outputDir/post_place_timing_summary.rpt #生成时序报告
#
# STEP#5: run the router, write the post-route design checkpoint, report the routing
# status, report timing, power, and DRC, and finally save the Verilog netlist.
#
route_design       #布线
write_checkpoint -force $outputDir/post_route.dcp #存档
report_route_status -file $outputDir/post_route_status.rpt #报告布线状况
report_timing_summary -file $outputDir/post_route_timing_summary.rpt    #生成时序报告
report_power -file $outputDir/post_route_power.rpt  #生成功耗报告
report_drc -file $outputDir/post_imp_drc.rpt        #运行DRC 生成DRC检查报告
# write_verilog -force $outputDir/cpu_impl_netlist.v -mode timesim -sdf_anno true
#
# STEP#6: generate a bitstream
#
write_bitstream -force $outputDir/SCRIPT_TEST.bit  #生成bit文件

##########################################################################
##########################################################################
##########################################################################
```

另外，在运行Vivado GUI的时候，工程文件的路径下会有一个.jou的文件，会自动记录所有GUI操作对应的Tcl脚本，便于查找与使用。

关于Tcl的学习，网上文章不少，这里只推荐xilinx的相关文档

UG892 讲述Vivado设计流程，non-project模式中有针对Tcl脚本的说明

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug892-vivado-design-flows-overview.pdf

UG894 讲述如何在Vivado中使用Tcl的文档

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug894-vivado-tcl-scripting.pdf

您将以匿名用户身份发表，请登录|注册

验证码:

发表