更好的管理 VIVADO 项目

最近在划水 FPGA 的一个培训，因为可以兑换学校的实习学分，所以就凑了热闹，不过重新温习这个东西我也有了一点对 Vivado 项目管理的想法，同时也找到了一些有用的东西

Introduction

主要的想法源自于 Vivado 编译项目时太过缓慢，GUI 经常失去响应的样子，十分笨重。Vivado 是可以利用 tcl 脚本来管理编译行为的，这个与 `make` 或者 `Ninja` 这类的软件类似，也就是说，在做一些项目的时候，掌握 tcl 脚本的写法可以完全不使用 Vivado 的笨重 IDE 直接使用 `vivado -mode batch / vivado -mode tcl` 来进行操作，又因为在 Vivado 中调用自己顺手的 Editor 实在费劲，字体也很奇怪，但是为了生产力，一定情况下是可以抛弃 GUI 界面的。当然啦，这一系列操作在 linux 下更方便 :p

Short Hand

大体来说，开发 FPGA 经历编写 RTL 代码，综合，布线，写入 2 进制 bit 文件，下板，也许之前还需要进行时序约束等步骤。而这一系列的工作，我认为编写一套顺手的 tcl 脚本来管理开发流程是十分有必要的，一方面可以让自己开发更加心中有数，另一方面减少了 GUI 消耗的内存和 CPU 资源。除此之外，网络上面很多开源的 RTL 代码是只有源代码的，以及像 chisel 编写的代码是只有 scala 源码的。怎么把这些工程整合到 Vivado 中呢？当然你可以一个一个文件的添加到 Vivado 工程中，然后点击综合，布线，生成 bit 文件，如果出错继续再点一遍上述的按钮。而我认为，编写一套合理的脚本帮我们自动化完成这些任务是很有必要的，这也是我转向 tcl 脚本管理项目的原因之一。

除此之外，Vivado 默认的代码路径十分不友好，比如：`/project.srcs/constrs/new/constr.xdc` 或者 `/project.srcs/rtl/new/top.v` 等等。你可以看出来，文件树太深，命名古怪，编程老手可不会这样（虽然我不是 :p ）！

Easy Tutorial

一下内容基本可以在 ug975, ug893 找得到。Vivado 在启动的时候有几个启动模式，分别为 `tcl`, `batch` 和 `gui`;

```
$ vivado -mode [ tcl | batch -source <your script> | gui ]
```

tcl

tcl 模式是进入了 vivado 提供的 tcl 命令行，我们的 tcl 脚本中的语句可以拆开来一条一条的输入到命令行中。这个是比较 interactive 的

batch

batch 模式就是一次执行一个 tcl 脚本

gui

gui 模式就是我们通常看到的图形界面，它的下面也有一个 tcl 命令行，但是 gui 实在太笨重了，我不会选择打开如此笨重的软件只为使用一个命令行

维护一个项目最简单的脚本可以这样写

Non-Project Mode

```
# set properties {

    set output_dir ./build
    file mkdir $output_dir

    set_part xc7z020clg400-1
# }

# read constraints and source files {
    read_verilog [ glob ./src/rtl/*.v ] # read all the verilog files
    # read_verilog ./src/rtl/top.v      # read single verilog file
    # read_verilog ./src/rtl/top.vhdl   # read single VHDL file

    read_xdc ./src/contrs/project.xdc
# }

# run synthesis {
    synth_design -top top    # Tell vivado your top module name

    # optional commands: checkpoints and reports
    # write_checkpoint -force $output_dir/post_synth
    # report_utilization -file $output_dir/post_synth_util.report
    # report_timing -sort_by group -max_paths 5 -path_type summary \
      -file $output_dir/post_synth/post_synth_timing_report
# }

# run placement and optimization {
    opt_design
    power_opt_design
    place_design
    phys_opt_design

    # optional commands: checkpoints and reports
    # write_checkpoint -force $output_dir/post_place
```

```
    # report_clock_utilization -file $output_dir/clock_util.report
    # report_utilization -file $output_dir/post_place_util.report
    # report_timing -sort_by group -max_paths 5 -path_type summary \
    #   -file $output_dir/post_place_timing.report
# }

# run routing {
    route_design

    # optional commands: checkpoints and reports
    # write_checkpoint -force $output_dir/post_route
    # report_timing_summary -file
$output_dir/post_route_timing_summary.report
    # report_utilization -file $output_dir/post_route_util.report
    # report_power -file $output_dir/post_route_power.report
    # report_methodology -file $output_dir/post_impl_checks.report
    # report_drc -file $output_dir/post_imp_drc.report

    # optional commands: write optimized, primitive based netlist
    # write_verilog -force $output_dir/netlist.v
# }

# write bit stream {
    write_bitstream $output_dir/top.bit
# }
```

Project Mode

Project Mode 与上面的脚本类似，只不过更加偏向 Vivado 自动生成一系列默认文件的模式

```
# create project and set device parts {
    create_project myproject ./myproject -part xc7z020clg400-1
# }

# add source files {
    add_files [ glob ./src/rtl/*.v ]
    import_files -force -norecurse
    import_files -fileset constrs_1 ./src/contrs/constrs.xdc
# }

# run synthesis, implementation, write bitstream {
    launch_runs synth_1 -to_step write_bitstream
    wait_on_run impl_1
# }
```

Idea

一点小小的想法，参照 cmake 的设计理念，我们是否可以开发一款叫做 hdlmake 的软件进行软件管理，可以在 Vivado 或者 alera 或者 yosys 这类综合软件之间切换，也可以针对不同的平台进行 tcl 脚本的自动生成，因为很多软件的行为在 Windows 和 Linux 下是十分不同的，比如路径问题（`/User/somebody` 和 `D:\\User\\Somebody`）。带着这个想法，结果发现其实有人做过了，而且项目名字就叫做 hdlmake，暂时还没有时间去品尝螃蟹，有人愿意尝试吗 :p