DOULOS

KnowHow

Tcl Scripting with Actel Designer

Tcl Implementation in Actel Designer

Actel Designer provides a Tcl implementation. This is invoked by using the Actel Tcl shell

```
acttclsh
```

On Windows, a convenient way to launch a script is to write a batch file containing a line like this:

```
acttclsh do_actel.tcl %1 %2 %3 %4 %5
```

Let's look in a bit more detail at the stages of design flow using Actel Designer, but controlled using Tcl. The following examples assume synthesis has been carried out as a separate step, so we are starting from a sythesized netlist.

Creating a project and starting design

First we will set some variables with basic information about our design and project.

```
set compile_directory     actel
set top_name              chip
set family                ProASIC3
set part                  A3P250
set package               "208 PQFP"
set pdc_filename          ../../source/board_support/actel.pdc
```

The basic use of Actel requires the tool designer to be launched with a Tcl script, like this:

```
  designer "SCRIPT:script.tcl arg1 arg2" "SCRIPT_DIR:path_to_script" "LOGFILE:file.log"
```

In the following examples, we need to supply a number of different scripts to the designer software, depending on what we are doing. Sounds like a job for a Tcl procedure!

```
  proc run_designer {message script} {
    puts "Designer: $message"
    set f [open designer.tcl w]
    puts $f $script
    close $f
    puts [exec designer SCRIPT:designer.tcl]
}
```

This procedure opens a file `designer.tcl`, and dumps a script into it. The script is supplied as the second argument to the procedure. Finally the script is executed and passed to designer (remember square brackets in Tcl `[]` return the result of evaluating an expression - in this case the result of the `exec` command).
Now with the aid of our little "helper" procedure we can start of a project:

```
run_designer "starting new project" "
  new_design \
    -name $top_name \
```

```
    -family $family \
    -path .
  set_device \
    -die $part \
    -package \"$package\" \
    -speed -2 \
    -voltage 1.5 \
    -iostd LVTTL \
    -jtag yes \
    -probe yes \
    -trst yes \
    -temprange COM \
    -voltrange COM
  import_source \
    -format edif \
    -edif_flavor GENERIC $top_name.edn \
    -format pdc \
    -abort_on_error yes $pdc_filename \
    -merge_physical no \
    -merge_timing yes
  save_design $top_name.adb
"
```

This uses Actel specific commands such as new_design, set_device, import_source, and save_design to create a project (note the source is in `edif` format, it is the output of a synthesis tool we ran earlier). The project database is stored in a `.adb` file. Note how the complete script is embedded in a Tcl string as the second argument to the run_designer Tcl proc. Note also the use of the backslash character `\` to indicate that a line is continued.

Compiling the design

To compile the design, we need to use the Actel Designer compile command, follwed by save_design to store the project. The various compile options (in fact the options for all the Actel Designer Tcl commands) are described in the Actel Designer help. Here's the script, again passed as an argument to the run_designer Tcl proc we wrote earlier.

```
run_designer "compile" "
  open_design $top_name.adb
  compile \
    -pdc_abort_on_error on \
    -pdc_eco_display_unmatched_objects off \
    -pdc_eco_max_warnings 10000 \
    -demote_globals off \
    -demote_globals_max_fanout 12 \
    -promote_globals off \
    -promote_globals_min_fanout 200 \
    -promote_globals_max_limit 0 \
    -localclock_max_shared_instances 12 \
    -localclock_buffer_tree_max_fanout 12 \
    -combine_register off \
    -delete_buffer_tree off \
    -delete_buffer_tree_max_fanout 12 \
    -report_high_fanout_nets_limit 10
  save_design $top_name.adb
"
```

Design Layout

Hopefully you can now see the pattern - we keep calling run_designer with different arguments. Here is the call and script for layout...

```
run_designer "layout" "
  open_design $top_name.adb
  layout \
    -timing_driven \
    -run_placer on \
```

```
    -place_incremental off \
    -run_router on \
    -route_incremental OFF \
    -placer_high_effort off
  save_design $top_name.adb
"
```

Exporting a programming file

Actel Designer uses a format called STAPL, which is read by the programming tool, `flashpro`. So next we need to create the STAPL file.

```
run_designer "exporting STAPL file" "
  open_design $top_name.adb
  export \
    -format bts_stp \
    -feature prog_fpga \
    $top_name.stp
  save_design $top_name.adb
"
```

Programing the device

Finally we need to call flashpro to programme the device. flashpro requires a control file in a tagged format that looks a bit like HTML or XML. So we need to create the control file, then invoke flashpro.
Here's the Tcl code to do that.

```
file mkdir work
cd work
file copy -force ../$top_name.stp copy.stp
set f [open $top_name.pro w]

puts $f "
<project name=\"$top_name\" version=\"1.0\">
  <ProjectDirectory>
    [pwd]
  </ProjectDirectory>
  <View>
      SingleSTAPLView
  </View>
  <LogFile>
    flashpro.log
  </LogFile>
  <programmer status=\"enable\" type=\"FlashPro3\" revision=\"UndefRev\" connection=\"usb2.0\">
  </programmer>
  <configuration>
    <Hardware>
      <FlashPro>
        <TCK>
          0
        </TCK>
        <Vpp/>
        <Vpn/>
        <Vddl/>
        <Vdd>
          2500
        </Vdd>
      </FlashPro>
      <FlashProLite>
        <TCK>
          0
        </TCK>
        <Vpp/>
```

```
        <Vpn/>
      </FlashProLite>
      <FlashPro3>
        <TCK>
          0
        </TCK>
        <Vpump/>
      </FlashPro3>
    </Hardware>
    <Algo type=\"STAPL\">
      <filename>
        $stapl_filename
      </filename>
      <local>
        copy.stp
      </local>
      <SelectedAction>
        PROGRAM
      </SelectedAction>
    </Algo>
  </configuration>
</project>
"
close $f

catch {exec flashpro.exe $top_name.pro}
  puts "Programming done"
} else {
  puts "flashpro.exe does not appear to be in the PATH environment variable."
}
cd ..
file delete -force work
```

This looks a bit more complicated than the earlier steps, but that's because we have to write out the control file. If you take out the control file, the code is really this:

```
file mkdir work
cd work
file copy -force ../$top_name.stp copy.stp
set f [open $top_name.pro w]

#write file here (omitted)

catch {exec flashpro.exe $top_name.pro}
  puts "Programming done"
} else {
  puts "flashpro.exe does not appear to be in the PATH environment variable."
}
cd ..
file delete -force work
```

which doesn't look so bad...

Conclusion

We've shown you

- How to run Actel Designer from the command line and pass it a script
- How a simple Tcl procedure can be used to encapsulate passing a script to Designer and running it.
- Creating a control file for `flashpro.exe` from Tcl

You'll find scripts like this used in our training course Comprehensive VHDL.

Back to top