By Joshua Cantrell
jjc@icsl.ucla.edu

Extended Synopsys Tutorial

Scripts

Introduction

The best way to use Synopsys is to use the menus as little as possible when first compiling your design. This helps avoid missing steps and strange outputs that you may not understand. For this reason, the first element of Synopsys to be covered will be the script file followed by using a few of the menus.

The Synopsys Initialization File - *.synopsys_dc.setup*

Synopsys has a file to store information on the default environment. These are placed in a file called *.synopsys_dc.setup.* When this is provided for you, it will be called *DOT.synopsys_dc.setup* so that you won't miss it. Simply rename this file to remove the *DOT* in the name, and place it in the same directory you run "design_analyzer". This will setup the library variables so you don't have to go to "*Setup / Defaults...*" whenever you start Synopsys.

Running a Script

First, the Synopsys Design Analyzer must be started by running "design_analyzer". Under the menu "*Setup*", select "*Execute Script...*". By first browsing to the correct directory, you can then either select or type in the name of the script file followed by pressing the *OK* button. The script files are usually appended with ".*scr*".

Setting Environment Variables in a Script

There are many environment variables that you may change often, but don't want to remember to change each time you run Synopsys. Adding lines like the following set the given variables to the desired values. Values found between braces, '{' and '}', represent a list of values.

```
/*
 * Setup Libraries.
 */
target_library = ee216a.db
symbol_library = generic.sdb
link_library = { ee216a.db }
```

A more complete list of the variables can be found by selecting the menu option "*Setup / Variables...*".[1]

Types of Values in Script

The types of values found in a script are dependent on the Synopsys library. The three major types of units are *time*, *capacitance*, and *resistance.* These could be nanoseconds, picofarads, and kiloohms. It is important to know these units so you can correctly setup and interpret compilation statistics.

---

[1] These variables are only used as an example. Please do not include these lines in your *.scr files.

Linking a Design

The three commands used to link a design are *read, current_design,* and *link*. *Read* tells Synopsys which design files it will need to load in to compile the design.[2] In the example below, only the file *traffic.v* is needed. The *current_design* is the module to be compiled, *traffic* in this case. The *link* command finds all of the refrenced library components and designs and links them to the current design.

```
read -f verilog { traffic.v }
current_design traffic
link
```

Defining the Clock

The *create_clock* command is used to specify a clock. It requires two arguments, the node name (*clock* in the example below), and the clock's period (notice this is in the time units of the library in use). Other conditions can be set on the clock by using the correct commands. One useful command is *set_clock_skew,* which allows you to indicate the uncertainty of the clock pattern. It's arguments are the time units of deviation from the nominal clock period and the effected clock nodes.

```
create_clock clock -period 5
set_clock_skew -uncertainty .25 clock
```

Defining the clock's period tells Synopsys what clock speed it should use when optimizing the design. Ordinarily, Synopsys will try to optimize for both speed and area. If a clock's period is large, Synopsys will make a trade off, decreasing area and sacrificing speed. If a clock's period is small, Synopsys will increase the area to gain speed.

Setting the Input Pins

The *set_driving_cell* command tells Synopsys what it should consider as inputs into the logic to be created and optimized. It's important to give it two arguments, the name of the cell driving the input (a minimum sized inverter in the example), and a list of the inputs being driven by this cell. Notice that a general variable (a list in this case) can be defined and used as the an argument.

```
input_pins = {reset, carSenseWes, carSenseWil}
set_driving_cell -cell stdinv_1x input_pins
```

---

[2] If Synopsys gives errors related to *Initial* statements or other unrealizeable elements in the Verilog code, you must first cut these out and then load the file.

Setting the Output Pins

Output pins can be specified using several commands. Some common ones are, *set_load*, *set_fanout_load*, and *set_output_delay*. The *set_load* command tells Synopsys what type of additional capacitance to hang at the given pins. For outputs, this corresponds to the output load. (I'm not sure what *set_fanout_load* does yet.) The *set_output_delay* command sets a delay relative to the given clock on the given nodes.

```
output_pins = {liteWes[2], liteWes[1], liteWes[0],\
               liteWil[2], liteWil[1], liteWil[0]}
set_load -pin_load 0.03 output_pins
set_fanout_load 0.03 output_pins
set_output_delay 0.5 -clock clock output_pins
```

Compiling and Optimizing the Design

A few commands are important to actually compile and optimize the design. *Uniquify* produces duplicates of modules which are used in more than one place. *Set_fix_hold* makes sure that any hold errors due to the given clocks are fixed. *Compile* tells Synopsys to do the actual compilation. There are three levels of effort for mapping a design, *low*, *medium*, and *high*. The highest takes the most time, but does the best job.

```
uniquify
set_fix_hold clock
compile -map_effort medium
```

Other commands you may want to look up are *set_structure*, *set_flatten*, *set_max_delay*, and *set_max_area*.

Saving the Result

To save the design and results, a *write* command must be given:

```
write -format db      -hierarchy -output traffic_gate.db
write -format verilog -hierarchy -output traffic_gate.v
write_timing -f sdf-v2.1 -context verilog -o traffic_gate.sdf
```

Saving Reports in a File

To save things in a file, a '>' and '>>' can be used. The single '>' creates a new file, and the '>>' appends to the given file.

```
report_timing > traffic.rep
report_area >> traffic.rep
report_wire_load >> traffic.rep
report_design >> traffic.rep
```

Telling Synopsys to Quit

If you don't need Synopsys to continue running when you are done with the script, you can tell it to exit with the *quit* command.

```
quit
```

Navigating Synopsys

Finding Help!

If you want to really learn what the scripting commands do, you can use the help in Synopsys. Use the menu option "*Help / Commands*". You can type in the *Command Name* followed by hitting the enter key, and the help will appear in the text scrollbox.

Decending into the Logic of a Module

You can get to the logic of a module after compilation by double clicking on it using the left mouse button. The little icon with a gate on it will blow up to show a module with inputs and outputs. Double click on the module, and it will blow up to show logic.

Visually Checking the Attributes of Logic

While in the window showing the logic of the desired component, you can see many things by highlighting paths. The many options can be seen by going to the submenu "*Analysis / Highlight*". The most important for you to get to know is for the critical path, "*Analysis / Highlight / Critical Path*".

To find out where wires go, you can click on them, and then the net name will appear at the bottom of the window, and the selected wire will become broken. Likewise, the names of the components used (both instance and library names) can be discovered by selecting them.

Printing a Design

To print the optimized output, select "*File / Plot...*".

Tweaking for Optimality

If Synopsys isn't good enough for you, some variables can be tweaked in the "*Attributes*" menu. To change the clock period without having to run a script, or type in a command, and select the clock's pin, select "*Attributes / Clocks / Specify...*". "*Tools / Design Optimization...*" can be used to recompile the design at a low, medium, or high effort.

Viewing Text Reports

Text reports can be found by selecting "*Analysis / Report...*". This brings up a window so you can select what types of reports you would like. Important ones are under *Analysis Reports* and are listed below:

- Timing - Critical path timing and slack time (wasted time with the given period).
- Point Timing - Select a wire, and this gives you a critical path containing the desired net.
- Area - The estimated area the logic will take up when placed and routed.

A useful feature of the *Report* window is that you don't need to close it. Simply select the *one* report you wish to see, then press the *Apply* button. When you want to see another, deselect the currently selected option, and select the one you wish to see.

Reading the Timing Reports

The *Timing* and *Point Timing* reports show a path and give timing values in that path. It is critical that you can understand how to read what is shown. A sample path is given below:

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : traffic
Version: 1999.10
Date   : Fri Nov 10 13:22:10 2000
****************************************

Operating Conditions: CDA_TYPICAL   Library: hp.4u3m1p_10_std_sd
Wire Load Model Mode: enclosed

  Startpoint: lite_state_s_reg[1]
              (rising edge-triggered flip-flop clocked by clock)
  Endpoint: lite_state_s_reg[1]
            (rising edge-triggered flip-flop clocked by clock)
  Path Group: clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  traffic            CDA_WIRE1             hp.4u3m1p_10_std_sd

  Point                                                   Incr       Path
  ------------------------------------------------------------------------
  clock clock (rise edge)                                 0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  lite_state_s_reg[1]/CLK (stdtgdff_q_2x)                  0.00       0.00 r
  lite_state_s_reg[1]/QBAR (stdtgdff_q_2x)                 0.71       0.71 f
  U14/Y (stdnand2_2x)                                     0.21       0.92 r
  U34/Y (stdinv_2x)                                       0.04       0.96 f
  U38/Y (stdor2_3x)                                       0.29       1.25 f
  U17/Y (stdinv_3x)                                       0.09       1.34 r
  U13/Y (stdnor2_1x)                                      0.10       1.45 f
  U32/Y (stdaoai211_1x)                                   0.32       1.77 r
  U31/Y (stdnand2_2x)                                     0.10       1.87 f
  U20/Y (stdand2_1x)                                      0.31       2.18 f
  U25/Y (stdoaoi211_1x)                                   0.30       2.48 r
  lite_state_s_reg[1]/D (stdtgdff_q_2x)                    0.00       2.48 r
  data arrival time                                                  2.48

  clock clock (rise edge)                                 5.00       5.00
  clock network delay (ideal)                             0.00       5.00
  clock uncertainty                                      -0.25       4.75
  lite_state_s_reg[1]/CLK (stdtgdff_q_2x)                  0.00       4.75 r
  library setup time                                     -0.05       4.70
  data required time                                                 4.70
  ------------------------------------------------------------------------
  data required time                                                 4.70
  data arrival time                                                 -2.48
  ------------------------------------------------------------------------

  slack (MET)                                                        2.22
```

On the upper left hand side of the timing table is the name *Point*. Below this are listed components in the circuit which a signal travels through. To the right of this is the name *Incr*. Below this are listed the delay associated with the components on the left. To the extreme right is the name *Path*. Below this is the total time the signal has spent in the path at a given point. To the right of the numbers is an *r* if the signal is rising, or *f* if the signal is falling.

The *data arrival time* represents the time it takes the signal to traverse the given path. In the *Timing* report, this is the critical path. Below this is timing related to the clock. The *data*

required time represents the minimum amount of time found between the rising and falling clock edges. The *slack* is the amount of wasted clocking time. It tells us that we could decrease the clock by the given amount, and the circuit would still operate properly.

Synopsys Directives in Verilog

Introduction

Synopsys allows users to add comments to Verilog code that tell it how to compile. These are directives for Synopsys even thought they look like comments in Verilog. This means the directives either follow `//` or are between `'/*'` and `'*/'`.

The *translate_off* and *translate_on* Directives

These directives are useful when you have someVerilog code that Synopsys cannot compile. An example of this would be an*initial* statement. Used, this may look like:

```
// synopsys translate_off
   initial
      begin
         a = 1'b0;
         b = 1'b0;
         #10
         a = 1'b0;
         b = 1'b1;
      end
// synopsys translate_on
```

where the Verilog code between both directives is ignored by the Synopsys compiler.

The *parallel_case* Directive

This tells Synopsys to interpret a *case* statement in Verilog in a particular way. Normally Synopsys produces a priority encoder from a *case* statement, but *parallel_case* tells Synopsys to generate multiplexer logic instead. For *parallel_case* to work properly, only one case item should be executed at a time. This directive must follow immediately after the *case* statement's keyword as shown below:

```
case (in) // synopsys parallel_case
   0: out = 2;
   1: out = 1;
   2: out = 3;
   3: out = 0;
endcase
```

The *full_case* Directive

This tells Synopsys that all possible inputs are covered by the given Verilog *case* statement, and that there doesn't need to be a *default* case. An example of *full_case* is shown below:

```
case (in) // synopsys full_case
   0: out = 2;
   1: out = 1;
   2: out = 3;
endcase
```

Because there is nothing given for when *in* is 3, using *full_case* tells Synopsys not to worry about that possible input, and treat it as though all the possible inputs were covered. If all possible inputs are covered, then the *full_case* directive is not necessary because Synopsys will already interpret it as if it were *full_case*.

Learning More About Design Analyzer

Check out the menu option "Help / On-Line Documentation... ". Look under "CORE SYNTHESIS TOOLS". There should be many topics that describe how the Design Analyzer works.