ECE 128 – Synopsys Tutorial: Using the Design Compiler
Created at GWU by Thomas Farmer
Updated at GWU by William Gibb, Spring 2010
Updated at GWU by Thomas Farmer, Spring 2011

Objectives:

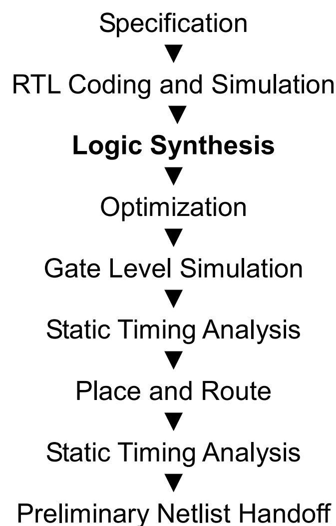
- Synthesize a “structural” 1-bit full adder using the Synopsys Design Compiler
- Synthesize a “behavioral” 1-bit full adder using the Synopsys Design Compiler
- Synthesize both full adders using the AMI .5 library using the OSU Standardized Cells

Assumptions:

- Student has completed lab 1 and has a working structural 2-bit full adder

Introduction:

The ASIC design flow is as follows:



In this tutorial, we will be working in “Logic Synthesis” portion of the ASIC flow. In this course, we will use the Synopsys Product Family for synthesis. IN particular, we will concentrate on the Synopsys Tool called the “Design Compiler.” The Design Compiler is the core synthesis engine of Synopsys synthesis product family.

It has 2 user interfaces :-
1) Design Vision- a GUI (Graphical User Interface)
2) dc_shell - a command line interface

In this tutorial we will take the verilog code you have written in lab 1 for a full adder and “synthesize” it into actual logic gates using the design compiler tool. We will use the GUI first, and after you become more familiar with the commands, you can migrate to dc_shell and drive the tool with scripts.

Part I: Basic Overview of Synthesis:

In synthesizing a design in Synopys' design compiler, there are 4 basic steps:

1) Analyze & Elaborate
2) Apply Constraints
3) Optimization & Compilation
4) Inspection of Results

Part II: Preparation

The preparation for running Design Compiler is a two part process, first you must create a settings file for the program (only once), and the must prepare your project (every time). Any time you wish to “synthesize” some verilog code, create a directory in your ece128 folder to house all of the files that will be created during the synthesis process. For this example, follow the instructions to setup your settings file and then prepare the directory structure:

1. Design Vision and DC_Shell both need to have setting loaded, to indicate to it where to look for library files and which libraries to work with. A comprehensive setup file has been crafted that you can use.

   Login to a workstation, open up a terminal window and type:

   ```
   ln –s /home/class/vlsi/course_ece128/env_files/.synopsys_dc.setup.S10
   .synosys_dc.setup
   ```

   This creates a symbolic link to a master .synopsys_dc.setup script that you can use for ECE128 this spring. It will contain search paths to all of the standard cell libraries that you need, as well as default settings to target the OSU standard cell library. For a more detailed discussion of the contents of .synopsys_dc.setup, see **Part VIII: Design Compiler setup file contents**.

2. Now you will need to setup your directory structure for this tutorial. Back in your terminal window type:

   ```
   cd ece128
   mkdir lab2
   cd lab2
   mkdir work
   mkdir src
   mkdir db
   mkdir reports
   ```

   Always create the “work” “src” “db” and “reports” directories in whatever directory you decide to work under. In our case, our 'working' directory will be 'lab2'

   Copy the AMI 0.5 “standard cell library's” verilog code into your “src” directory:

   ```
   cp /apps/design_kits/osu_stdcells_v2p7/cadence/lib/ami05/lib/osu05_stdcells.v  ~/ece128/lab2/src
   ```

3. Copy the verilog code you wish to synthesize into the “src” subdirectory:

   In this lab, we want to use the fulladder & halfadder you created in lab1:

   You may need to change the first part of the directory location to where your fulladder code is:

   ```
   cp ~/ece128/lab1/fulladder.v ~/ece128/lab2/src
   cp ~/ece128/lab1/halfadder.v ~/ece128/lab2/src
   cp ~/ece128/lab1/fulladder_tb.v ~/ece128/lab2/src
   ```

   The above three lines copy the “fulladder” “halfadder” and the test bench you created in lab1 into 'src' directory underneath the lab2 directory you created in step 2.

Part III: Starting The Design Compiler, Analyzing and Elaborating your Design

All of the verilog code that you wish to 'synthesize' should now be under the 'src' directory in your 'working' directory. In this tutorial: ~/ece128/lab2/src

Note: Before ever attempting to "synthesize" verilog code, you must ensure that it compiles properly (using a verilog simulator) and its waveforms are as you expect (using simvision). ***Only*** *after the code passes the simulation phase can you move on to the synthesis phase of the ASIC design flow.*

1. In a terminal window, login to hobbes

   `ssh -X hobbes` (accept any warnings, and type in your seas pw when prompted)

   Note: For the spring 2010 semester, synopsys tools are only available on hobbes, in future semesters this step may not be necessary

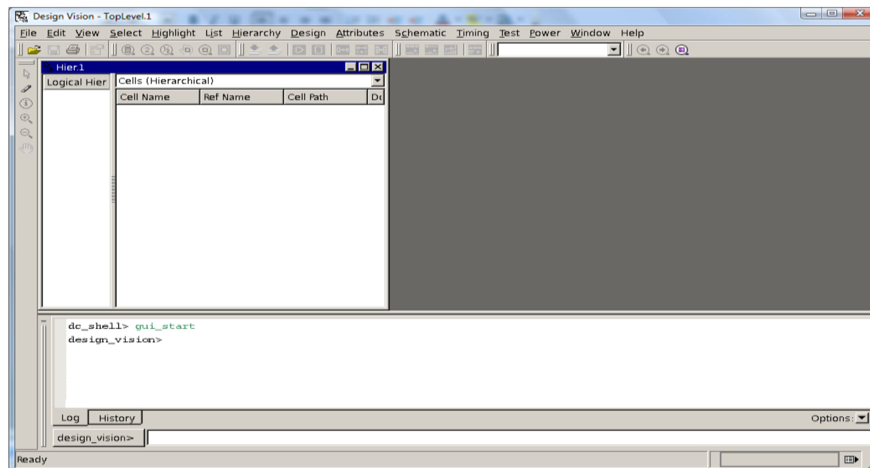
2. Change to your working directory

   `cd ~/ece128/lab2`

3. Start the design compiler's GUI by typing

   `design_vision` *(note: do NOT put an "&" after this command, it needs to run in the foreground)*

   Both the "dc_shell" and it's GUI will pop up and look something like this:



4) Click on **File**->**Setup** to verify that the parameters setup in the "synopsys_dc.setup" file have taken effect (look through both tabs to see all the variables and parameters setup). You should see the following in your screen:

   - Link Library: osu05_stdcells.db
   - Target Library: osu05_stdcells.db
   - Symbol Library: generic.sdb
   - Synthetic Library: dw_foundation.sldb

5) Load all your verilog code (and its dependent files) by going to: **File->Analyze**

Click on the “add” button and click on the “src” sub-directory
Add “fulladder.v” and “halfadder.v”

*Note : The **analyze** command will do syntax checking and create intermediate .syn files which will be stored in the directory work, the defined design library. The elaborate command goes to the work directory to pick up the .syn files and builds the design up to the Design Compiler memory.*

6) Inspect the messages in the LOG window at the bottom, correct any syntax errors in your verilog files and do the analyze again, otherwise, proceed.

Once you've succesfully analyzed the code, Select **File->Elaborate.** Your design is now translated to a technology independent RTL model.

Elaboration brings all the associated lower level blocks into the Design Compiler automatically (by following the dependency of the instantiations within the HDL code)

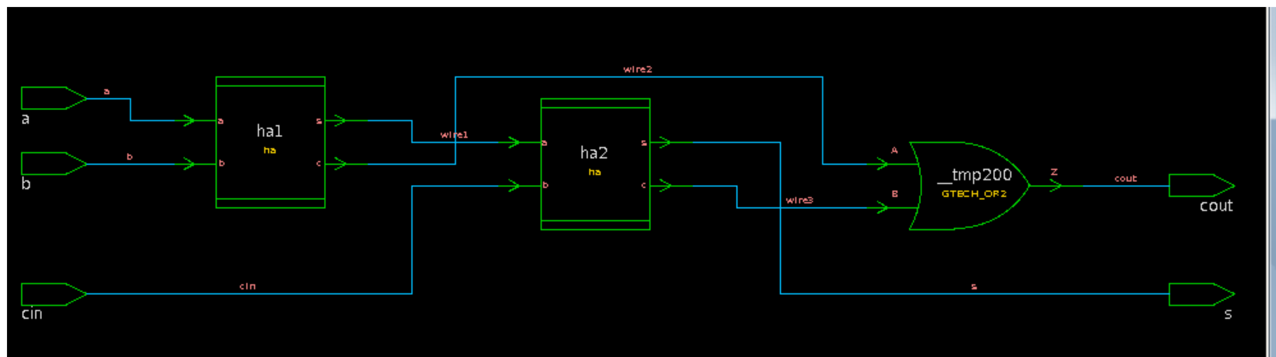
Instead of doing Analyze & Elaborate, you can also do just Read for a verilog design, the difference is that you have a choice of design library to place the analyzed design when you do Analyze, whereas with Read only the default library WORK is used.

You may see see warnings about the following: “The File ‘/apps/synopsys/B-2008.09/synthesis/dw/dw5/… is not a symb file.”. You can ignore these warnings.

Part IV: Viewing the schematic

You should notice your full adder and the half adder that it depends on is now loaded into the design compiler. The logical heirarchy shows

1) Click on **Schematic** → **New Design Schematic View** and the following schematic should appear



2) You can click on the “ha1” or “ha2” components to drill down further into the elaborated design.

As you drill down, various 'tabs' will open up. You can also use the icon on top of the menu bar to go back “up” in the design hierarchy

Notice the basic gates, the OR gates, have the label “GTECH” this stands for 'generic' technology, meaning the OR gates have no timing, power, or other realistic information contained within them. They are merely symbols.

Part V: Applying Contraints to your design

Adding constraints to your design is a process to make your design a bit more realistic than just simple gates. As an example, the wires that connect your gates are ideal, no R,L, or C. You can apply what is known as a 'wire model' to make the wires take on realistic RLC characteristics as they would in an extracted layout. Or another example would be to apply a 'fanout' or 'fanin' to the inputs and outputs of your design as to simulate a realistic level of input or output driving.

1. The first step is to "LINK" your design, click **File → Link Design**

   The link command checks to make sure all parts in the current design are available in Design_Analyzer's memory.

2. In the "Hierarchy" window, click on the fulladder (the top most portion of your design).

   Then from the file menu click on **Hierarchy → Uniquify → Hierarchy**

   Choose the top most level of your design (in this case "halfadder") in cell reference from the drop down list menu that comes up. Do not adjust any other defaults, and press "OK" to allow the uniqufying process to begin.

   The command uniquify is used to make unique the instances of the same reference design during synthesis.

   ***Q:** Why uniquify your design?*

   ***A:** When 2 instances of the same reference design/cell are present in the design (like how the "halfadder" module is used twice in our fulladder" design, then different constraints will have to be applied to the design; thus to resolve the multiple instances issue we use uniquify command.*

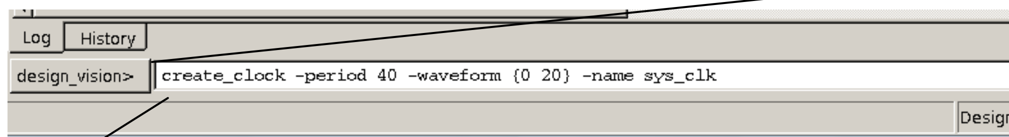
   If you were to now look at your schematic view, you'd notice that ha1 and ha2 have different internal names. They now have the names: ha_1 and ha_0. Before they both had the same name: ha...this step allows each half adder to be its own 'unique' module. We can now apply different 'constraints' to each half adder, without affecting the other instance of it in the design.

3. Setting up the clock

   a) If your design DOESN'T have a clock, follow this step. If your design DOES have a clock skip this step, go to the next step

   If you're design doesn't have a clock at all, then it is purely 'combinational' logic. We need to create what is known as a 'virtual' clock. The virtual clock would be analagous to a system clock that all of the signals in your design would be 'measured' against.

   To create a virtual clock named "clk" type the command below into the design_vision field:

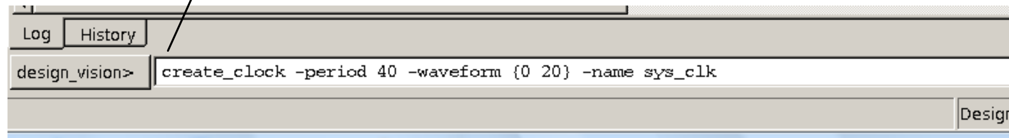



```
create_clock -period 40 -waveform {0 20} -name clk
```

   This creates a virtual clock with a 40 ns period, 50% duty cycle = 25MHz clock

b) if your design DOES have a clock pin, follow this step. If it does not, skip this step, move onto the next.

If your design is not purely combinational, and has some synchronous components (e.g. registers), you must create a clock with the same name for the clock's pin that you used in your verilog code. In the design_vision command window:



Type, this command:

```
create_clock clk -period 40 -waveform {0 20}
```

This creates a virtual clock with a 40 ns period, 50% duty cycle = 25MHz clock. This clock will be used to constraint the data paths between flip flops in your design.

c) Setting a 'delay' on the clock:

By default, Design Compiler assumes that clock networks have no delay (ideal clocks). Realistically, depending on the size of your design, the clock will have some 'skew' as it propogates through the system. To model this skew, type the following command in the design_vision command window:

```
set_clock_latency 0.3 clk
```

This will give the clk a .3ns skew in your design.

4. Setting up constraints on your input and output pins

- By default, the Design Compiler will model your system as if the signal arrives at the input ports at time 0. In most cases, input signals arrive at staggered times. Type the following command in the design_vision command window:

  ```
  set_input_delay 2.0 -clock clk [all_inputs]
  ```

  This 'input_delay' will assume that your design is driven by the slowest DFF in the AMI 0.5 library. The DFF has a clock-Q delay of 1.75ns. We add another 0.25ns for wiring delay. The overall delay will be 2ns 'relative' to the system clock that you setup in step 3.

- Assume that your design is driving a DFF where the DFF has a setup time of 1.4ns and another 0.25ns is for wiring delay. This command will set an 'output' delay on all the output pins of 1.65ns relative to the system clock:

  ```
  set_output_delay 1.65 -clock clk [all_outputs]
  ```

- The following three commands will set realistic 'loads' on each output pin. The second and third command sets 'maximum' fanin and fan-out for the input and output pins of your design. Type the following three commands in the design_vision command windows:

  ```
  set_load 0.1 [all_outputs]
  set_max_fanout 1 [all_inputs]
  set_fanout_load 8 [all_outputs]
  ```

5. To set the overall set of constrains on all of your input and output ports, type the following command:

```
report_port
```

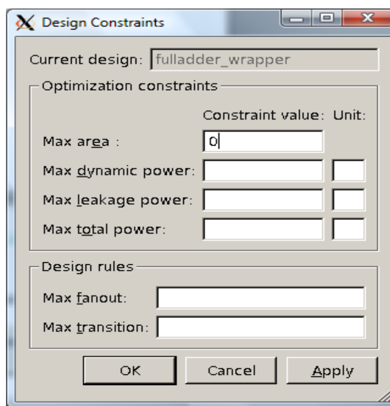
- This command will display each input and output port and show the constraints you have set on them. Check to ensure that each port has the constraints that you have set in the last 4 steps.

6. AREA vs. SPEED

- When the design_compiler synthesizes your design, it can attempt to minimize the 'area' of your design, and sacrifice speed. Or it can attempt to maximize the speed of your design by sacrificing area.
- To ask the design_compiler to synthesize for area, from the menu choose:

Attributes->Optimization Constraints->Design Constraints

Set the 'maximum area' to 0. This will force the synthesizer to optimize for the smallest possible area:



- If you wish the design compiler to optimize for speed, clear the max area field and press OK

7. Once you are finished specifying all of the design constraings, from the menu choose:

Design → Check Design, press **OK**

- This step will check your design's netlist description for problems like connectivity, shorts, opens, multiple instantiations. If your design passes this step, in the “LOG” window, you should see no errors.

8. You are now ready to have the design_compiler synthesize your design using the AMI 0.5 standard cell library.

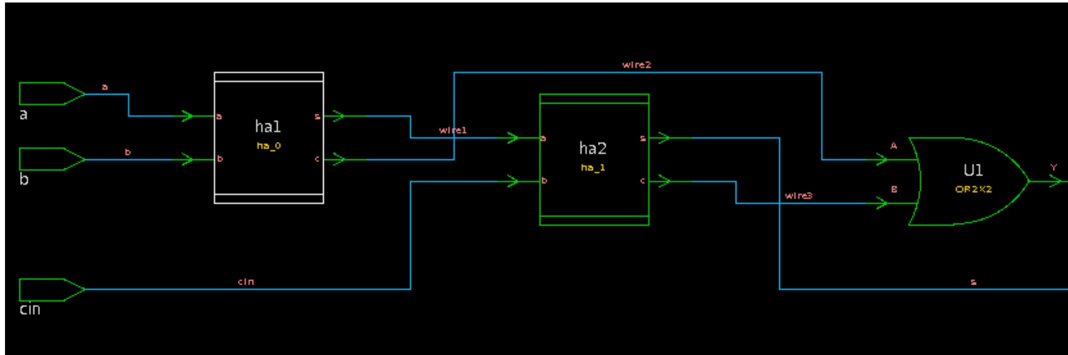
- From the menu choose: **Design → Compile Design**. You can ask the synthesizer to use 'low' 'medium' or 'high' effort to achieve the constraints you have set. The higher the effort, the longer the synthesize will take. If you design is large, this could mean several hours to days if you choose a 'high' effort.
- For this exercise, 'medium' effort will be sufficient, that should be the default, press OK to being synthesis

Part VI: Inspecting your Results

1. Once the synthesis is complete, you can view your 'synthesized' schematic. From the menu choose:

   Schematic → New Design Schematic View

   - Your design will now be implemented using the AMI 0.5 standard cell library.



   - You may double click on the 'half adders' to see what is inside each of them. You should see that they have been implemented using GATES from the standard cell library.
   - If you see components that don't map to a standardized cell, you may have done something in verilog that cannot be synthesized. You must investigate the cell and your verilog code to see what you have done incorrectly in verilog.

2. To view the critical path in your design, while in the schematic view, from the menu choose:

   Select → Paths From/Through/to
   set the: delay type: **max**

   - The critical path will then be highlighted on the schematic window using a white line.
   - Alternatively, you can select **Schematic → Paths From/Through/to**, and set the delay type to **Max**. A new window will open up showing the entire critical path, gate by gate.

3. Checking if your constraints have been met.

   - If you have setup a clock that is FAR to fast for your design, the design_compiler will synthesize something, but it won't have met the constraint you have setup. Or if you set a fanin/fanout load that is unreasonable, the same will occur. You need to check the results of the synthesis by checking the synthesis reports.

   - **Checking the Timing**

     - From the menu, choose: **Timing → Report Timing Path**
     - If you wish, you can have the design_compiler write this 'report' to a file, as well as the the screen. From the bottom of the dialog box, press: “To file:”
     - Then press OK

       - A 'report' will be generated and printed the screen. It will show the timing for each cell in the design as well as the entire design itself. If you have a clock in the system, it will show you if your design has 'met' the clock constraint, or 'failed' the clock constraint.
       - This is a crucial report to check. If you have violated the timing requirements, you may need to change the period of your clock, and then re-compile the design.

- **Checking the Area**

    - From the menu, choose: **Design → Report Area**
    - If you wish, you can have the design_compiler write this 'report' to a file, as well as the the screen. From the bottom of the dialog box, press: “To file:”
    - Then press OK
        - A 'report' will be printed on the screen showing the area consumed by each component. The area is in square micro-meters (um^2)

    - From the menu, choose: **Design → Report Cells**
        - This report will show the area consumed for each AMI 0.5 standard cell, it is a bit more detailed than the 'report area' results

- **Checking the Power**

    - From the menu, choose **Design → Report Power**
    - If you wish, you can have the design_compiler write this 'report' to a file, as well as the the screen. From the bottom of the dialog box, press: “To file:”
    - Then press OK
        - A 'report' will be printed on the screen showing the estimated power consumed by the design, both the switching and leakage power are shown.

Part VII: Re-simulating the synthesized design

- This next step is a crucial one. Now that your design has been synthesized, and your constaints have been met, it is time to use your verilog test bench (for the fulladder) against the design synthesized by the design compiler.

1. From the menu, choose: File->Save As
    - Navigate to your 'src' subfolder
    - Type in the name: fulladder_syn.v
    - Be sure not to call the file the same name as your original verilog file. If you do, it will overwrite your original work!

2. You may now exit the design_vision compiler

3. In a terminal window (on the local workstation, NOT on hobbes), change into the 'src' directory and type:

```
Sim-nc osu05_stdcells.v fulladder_tb.v fulladder_syn.v
```

This will use verilog to 'simulate' using your previously written 'test bench' fulladder_tb.v with the synthesized verilog code from the design_compiler: fulladder_syn.v

Open simvision and ensure the waveforms are correct. If they are not something has gone wrong during the sythesis. If they have gone right, you should now see 'delays' in your designs, representing the realistic timing characteristics of the AMI 0.5 standard cell library and the constraints you setup during the design_compiler session.

Part VIII: Design Compiler setup file contents

If you are curious about the contents of the DC setup file, you can view the file and read its comments. Type the following into a terminal to display the file contents onscreen:

```
Less ~/.synopsys_dc.setup
```

The default search_path is everything between double quotes "{".", "/opt/synopsys-3.5a/libraries/syn"}", this tells the Design Compiler to search for files or db at the current directory and at the libraries/syn directory where all the vendor libraries sources and db are placed. Instead of using class.db as your library, you can navigate to that libraries/syn directory, choose your preferred technology library and replace the above library assignment.

The link_library is used to define any technology input to the synthesis process, the "*" is necessary as it tells the Design Compiler to search for the existing databases in the Design Compiler memory first.

The target_library is the technology library to which you map your design during optimization.

A design library is a logical name referring to a UNIX directory which will store the intermediate files (the .mra .sim ... files) produced by analyze so as to not clutter up your present directory. You can choose other descriptive name besides work.

If you would like to setup and maintain your own .synopsys_dc.setup file, you can. Please consult with your TA before doing that.

The constraints file, as of 01/31/10, is included below.

```
#
# .synopsys_dc.setup file for ECE 128 - to synthesize to AMI .5 using OSU standard cell library
# created for Synopsys verison: B-2008.09, tjf 2/24/09
# Modified by William Gibb, 1/31/10
#

# user must create sub-directories: 'src' 'lib' 'work' and 'reports' for use with this setup file
set company {George Washington University}

#####
# Set search_path
#
# List locations where your standard cell libraries may be located
#
#####
set search_path [list . $search_path /apps/synopsys/B-2008.09/synthesis/libraries/syn/gtech.db /home/clas
s/vlsi/course_all/UofU/UofU_Digital_v1_2_oa3 /apps/design_kits/osu_stdcells_v2p7/synopsys/lib/ami05 "./sr
c" "./db"]

######
# Set Target Library
#
# Set a default target library for Design Compiler to target when compiling a design
#
######
# uncomment this if you are only doing syntheiss without scan chain (DEFAULT)
set target_library osu05_stdcells.db
# uncomment this if you are only doing synthesis against the UofU cells
#set target_library UofU_Digital_v1_2.db


#####
# Set Link Library
#
# Set a default library to link your design against.
#
#####
# uncomment this if you want to link against osu standard cells (DEFAULT)
set link_library { * osu05_stdcells.db }
# uncomment this if you want to link against UofU standard cells
```

```
#set link_library { * UofU_Digital_v1_2.db }
# uncomment this if ls /you want to link agaisnt lsi_10k.db
#set link_library {* lsi_10k.db}
# umcomment this if you want to work agaisnt gtech only
#set link_library { * gtech.db }


####
# Set Synthetic library
####

set synthetic_library [list dw_foundation.sldb]

####
# Set Schematic library
####

set symbol_library [list generic.sdb]

######
# MISC
######

define_design_lib work -path work;

#
# From UoUt
#
#/* ================================================= */
#/* General configuration settings.                   */
#/* ================================================= */

# Warn if latches are inferred
set hdlin_check_no_latch true

# Treat text between translate statements as comments
set hdlin_translate_off_skip_text true

#
set verilogout_write_components true

# Determines the name that will be used for the architecture of the write -f verilog command
set verilogout_architecture_name "structural"

# Turn tri state nets from "tri" to "wire"
set verilogout_no_tri true

# Treat text between translate statements as comments
set hdlin_translate_off_skip_text true

# List of package commands
set vhdlout_use_packages [list IEEE.std_logic_1164.ALL]

# Write out component declarations for cells mapped to a technology library.
set vhdlout_write_components true

# Determines the name that will be used for the architecture of the write -f vhdl command
set vhdlout_architecture_name "structural"


# Treat text between translate statements as comments
set hdlin_translate_off_skip_text true

# Specify the styel to use in naming an individual port member
set bus_naming_style {%s[%d]}
```

Part IX: Lab Assignment

Note: *Prior to performing the homework, the student should perform the tutorial first. If the student does not have a full adder or full adder test bench to use for this tutorial, they can copy one from the vlsi account. Type the following commands into your terminal after creating the folder structure in* ***Part II, step 2.***

```
cp ~vlsi/course_ece128/lab_files/lab2/fa_tb.v     ~/ece128/lab2/src/fa_tb.v
cp ~vlsi/course_ece128/lab_files/lab2/fa.v        ~/ece128/lab2/src/fa.v
cp ~vlsi/course_ece128/lab_files/lab2/ha.v        ~/ece128/lab2/src/ha.v
```

Lab Assignment Assumptions:

- Student has successfully completed Lab 1 and complete the above tutorial.
- Student has successfully completed Homework 1 or is working on Homework 1.

For ECE 128 students, your lab assignment is as follows:

1) Create a behavioral half adder, re-synthesize the full adder using this half adder, show results to TA
    a. Does the synthesize the same gates as your structural halfadder?
2) Synthesize your 2 bit full adder from Lab 1,
3) Synthesize your procedural DFF from Homework 1.

You will want to create a new directory to hold each part of this assignment, similar to what you did in **Part II, step 2** of the tutorial. Additional reports not highlighted in the lab, but listed in the deliverables section, may be found under the **Design** menu.

Deliverables due at the beginning of next lab:

- 2 Bit full adder, synthesized (3 pts)
- 2 Bit full adder synthesis reports (6 pts)
    - report design
    - report area
    - report cell
    - report port
    - report power
    - report timing
- Correct Simulation of the 2 bit full adder, synthesized, in your testbench due for lab 2. Include nc.log file. (3 pts)
- Dff, procedural synthesized (3 pts)
- Dff, procedural synthesis reports (6 pts)
    - report design
    - report area
    - report cell
    - report port
    - report power
    - report timing
- Correct simulation of your flip flop, synthesized. (3 pts). Include nc.log file.
- Write up. (5 pts)
    - Brief outline of what new processes you learned in this lab.
    - Identify the area required to implement your 2 bit full adder and the area required to implement your DFF.
        - Extra credit: Compare this area with an equivalent standard cell design you may have designed in ECE 126 (1pt)
    - Identify the power consumed by your 2 bit full adder and by your DFF.
    - Identify critical paths in your adder and DFF.
    - Results of synthesis and simulation for each module. Be sure to explain any odd behavior exhibited in your simulation.
- **Be sure to include the complete source code for all modules that you simulate, synthesize, or P&R, and any tool scripts used, for any assignment turned in for grading.**