ECE 6213 – Synopsys Tutorial: Using the Design Compiler

Prof. Jerry Wu. 2018

Introduction:

The ASIC design flow is as follows:

Specification
▼
RTL Coding and Simulation
▼
Logic Synthesis
▼
Optimization
▼
Gate Level Simulation
▼
Static Timing Analysis
▼
Place and Route
▼
Static Timing Analysis
▼
Preliminary Netlist Handoff

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. 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 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.

Note: This tutorial covered all what we discussed in the class. If you found any parameter is missing or the value is not what you expect, please modify it and indicate on the "discussion" of your report as a bonus.

Part I: Basic Overview of Synthesis:

In sythensizing 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

Any time you wish to "synthesize" some verilog code, create a directory in your ECE6213 folder to house all of the files that will be created during the synthesise process. For this **example**, prepare the following directory structure:

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

```
cd ECE6213          : you need to create your own ECE6213 directory first.
mkdir lab_DC        : project name; eventually, you will have a directory for final
                      project.
cd lab_DC
mkdir work          : your work library
mkdir src           : put all your source code
mkdir db            : put the db files
```

   Always create the "work" "src" and "db" directories in whatever directory you decide to work under.

   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  ~/ECE6213/lab_DC/src
```

2. Copy the verilog code you wish to synthesize into the "src" subdirectory:

   In this lab, we want to use the FIFO RTL you created in ECE6213 HW3:

```
cp ~/your_directory/your_FIFO.v ~/ECE6213/lab_DC/src
cp ~/your_directory/your_FIFO_testbench.v ~/ECE6213/lab_DC/src
```

   The above three lines copy the FIFO you created in HW3 into 'src' directory underneath the lab_DC directory you created in step1.

3. Create a file called: .synopsys_dc.setup in your home directory (this only needs to be done one time, you will not need to do it again if you have done in ECE6250/4150)

```
gedit ~/.synopsys_dc.setup
```

   Copy and paste the following lines:

```
set search_path [list . ${search_path} /apps/design_kits/osu_stdcells_v2p7/synopsys/lib/ami05 "./src" "./db"]
set target_library osu05_stdcells.db
set link_library { * osu05_stdcells.db }
set acs_work_dir "./work"
```

   This file should be **only 4 lines long** (the first line has "**wrapped**" in this tutorial)

Where:

- 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.

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: ~/ECE6213/lab_DC/src

*Note: Before ever attempting to "synthesize" verilog code, you **must ensure that it compiles properly** (using the 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 **linux machine (TBD in class; you may skip this step as long as step-3 works)**

   `ssh -X lnxrt01.cflab.seas.gwu.edu` (accept any warnings, and type in your seas pw when prompted; the exactly machine name to be discussed in class)

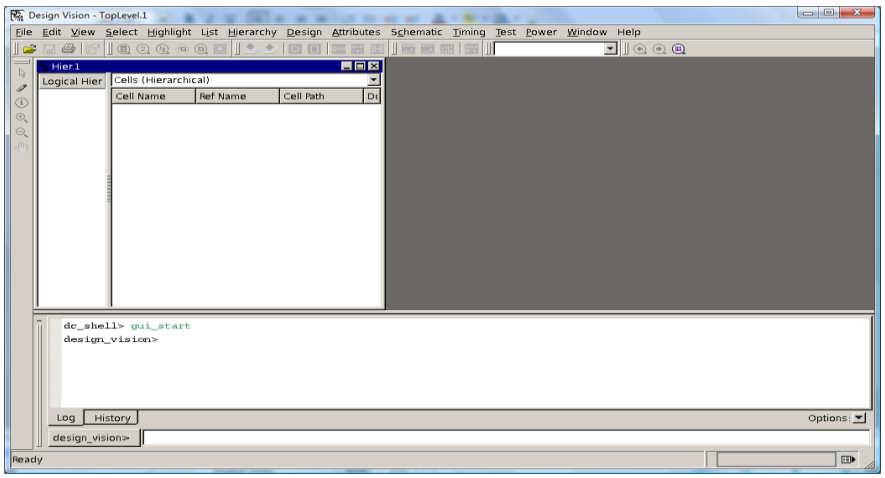
2. Change to your working directory

   ```
   cd ~/ECE6213/lab_DC
   ```

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 you setup in your "synopsys_dc.setup" file have taken effect (look through both tabs to see all the variables and parameters you've setup)

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 “your FIFO verilog files”

*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.

7) Once you've succesfully analyzed the code, Select File->**Elaborate**

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.

Your design is now translated to a technology independent RTL model.

Part IV: Viewing the schematic

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

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

2) You can click on the components name 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
Note that if your RTL design doesn’t include hierarchy, you will not see this feature/icon activated.

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.

In the “Heirachy” window, click on the top most portion of your design. (once again, if your RTL design doesn’t include hierarchy, you will not see this feature/icon activated.

2. If your design has hierarchy, do following step. Otherwise just understand why you need uniquify.

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

Choose the top most level of your design 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.

why uniquify your design?

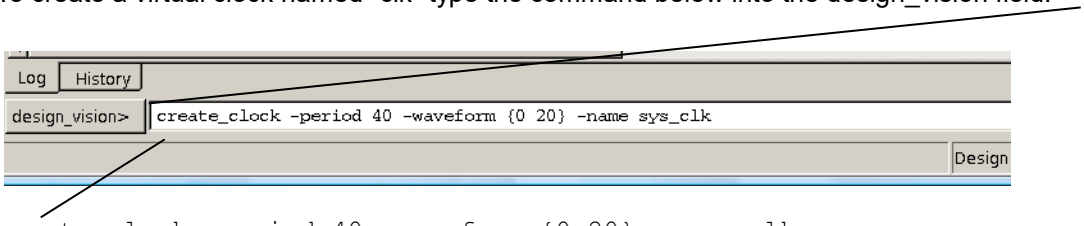
when 2 instances of the same reference design/cell are present in the design.

3. Setting up the clock **(Your clock name may be different from “clk” in this example.)**

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:



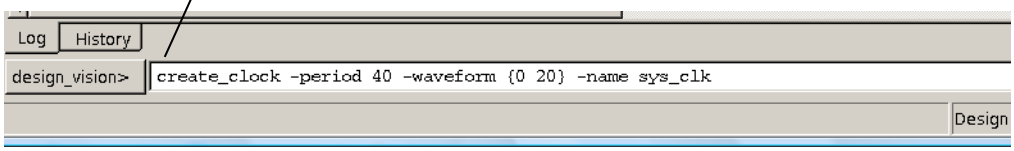

```
create_clock -period 40 -waveform {0 20} -name clk
```

(Your clock name may be different from “clk” in this example.)

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}
```

(Your clock name may be different from “clk” in this example.)

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

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 constrains 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_fan  out [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
```

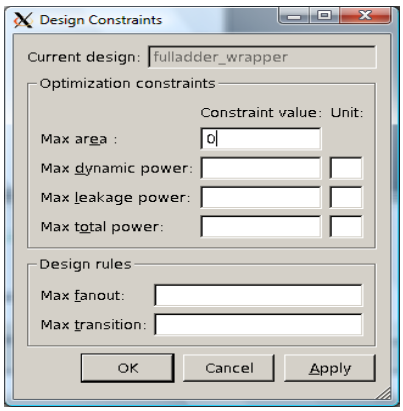
- This command will display each input and output port and show the constraints you have set on them. Check to ensure that each port has the constraints that you have set in the last 4 steps.

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 your schematic components 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.

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

Part VII: Re-simulating the synthesized design **- Part of your Home Work-6**

- 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 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: FIFO_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:

    ```
    verilog osu05_stdcells.v your_FIFO_testbench.v FIFO_syn.v
    ```
    (Use your FIFO testbench file in your Homework)

- This will use verilog to 'simulate' using your previously written 'test bench' your_FIFO_tb.v with the synthesized verilog code from the design_compiler: FIFO_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 characterstics of the AMI 0.5 standard cell library and the constraints you setup during the design_compiler session.
- You will need to do the same process on your final project.
- Please do the detail report of this tutorial as your homework assignment.
- Your report should at least included:
    - Timeing, Area report.
    - Critical timing identification.
    - Your synthesis results
    - Gate-level simulation with delay, as you expected, based on your timing constraints.
    - Discussion as need.

Home Work-6 Bonus points (5 pts)

- Find out two worst timing paths in your design (said path-1 is longer than path-2).
- Edit your gate level file and adjust the path-2 to be longer than path-1. (prove by new timing report)
- Please clearly specify this bonus work in your HW, so that the Grader can be aware.