Synthesis and APR Flow for EECS 427

This tutorial outlines a synthesis and auto-place and route (APR) design flow which will be used to design your program counter (PC), the controller modules, and a number of extra features / IO devices for your project. The flow will be partitioned into two main sections: (i) Synthesis and (ii) APR. This document describes the standard cell synthesis and automatic place and route (APR) design flow for use in the design of your controller and peripheral blocks.

Synthesis: Design Compiler is used to compile a synthesizable Verilog design into a gate-level structural verilog netlist containing instantiations of standard-cells obtained from a library. The user will provide a target clock frequency and input output constraints (timing and load capacitance) for each block in the design. Based in the timing properties of gates available in the standard-cell library (.lib format), Design Compiler produces a gate-level netlist that implements the desired functionality.

APR: Auto place and route is the process whereby a gate-level netlist (usually obtained from the synthesis tool) is physically implemented in layout by placing standard-cell layout and auto-routing the cells based on the connections inferred from the netlist. The routed design is then exported from the APR tool in a DEF (Design Exchange Format, more later) and subsequently imported by the Virtuoso Custom Layout tool.

Standard Cells: In this course, we will be using the 0.25micron, NCSU compatible VTVT (Virginia Tech VLSI for Telecommunications) standard cell library. The standard cell library contains schematic, symbol and layout views for various CMOS logic gates with a variety of drive strengths.

Setup

1. Find a nice place to do the tutorial (like your personal eecs427 directory)
2. Untar the tutorial file using:
   tar –xvf $CDK_DIR/../synth_tut.tar
3. A directory synth_tut will be created that will contain subdirectories synthesis and apr.

Synthesis

1. Change to the synthesis subdirectory of the tutorial
2. Open the synthesizable verilog design file dw_adder.v. We will be building a 16 bit adder using the DesignWare library.
3. Notice the parameter declaration in the verilog file and how it is used to determine the width of the desired adder when instantiating the DesignWare component. Other components that may be obtained from DesignWare are listed in the DesignWare manual. On your command prompt enter “sold” (Synopsys OnLine Documentation) and select DesignWare for more details.
4. Close the verilog and open the design compiler script dw_adder.tcl.
5. Note that the first thing in dw_adder.tcl is a command that sources dc_setup.tcl (in the class directory). dc_setup.tcl contains default design preferences as well as technology specific information (eg. Bus naming conventions, location of the timing library for the standard cell library used). The rest of the dw_adder.tcl file contains information that is specific to the adder design. It contains information that is specific to the design being implemented.
6. In this example the dw_adder that we have instantiated does not contain pipeline registers and therefore no real clock drives any nodes in the design. However, we still define a virtual clock clk, in order to provide a timing reference for the input and output delays. The input delay indicates how long after the rising clock signal (real or virtual) data is expected to arrive at the module inputs. The output delay indicates how long before the positive edge of the clock the outputs of the block are expected to arrive. Please see comments in the file that elaborate on different ways to specify input and output delays. This will be necessary when you design your modules.
7. Now that you have browsed through the files, you are ready to perform synthesis. There are two tools you can use to do this. Both use tcl command files. dc_shell-t is command line only whereas

design_vision has a gui that allows you to see schematics of the synthesized designs. Enter "dc_shell-t" or design_vision to start the respective tools. Simply enter the commands you see in the dw_adder.tcl file into the prompt (of the terminal window in both cases) or cut and paste the commands. (Later on of course, you can simply enter "source dw_adder.tcl" or add –f dw_adder.tcl on the command line when you invoke the tool to run your script. In this tutorial, for the benefit of familiarization, you should probably enter the commands one at a time.

8. In addition to the direct text outputs from the commands (which is where errors will appear) there are a few files that are created as results. Look through these to get used to what they are. (they are described below)

rep_file

Contains information about your design dw_adder, including area, power, as well as a timing report of the critical paths in the design. Make sure you understand how to read the delay path. It will prove very useful in producing good synthesized netlists.

netlist

syn_dw_adder.v, contains the gate-level netlist of your 16-bit adder. This netlist is used to provide connectivity information for the APR design flow.

SDF

syn_sw_adder.sdf, is a Standard Delay Format (SDF) file that contains simplified delays in your verilog netlist. Each gate in the verilog netlist specified by syn_dw_adder.v, is indicated in this file along with the associated gate delays from each input to each output. This sdf can be used in digital simulation by adding the following line to syn_dw_adder.v inside the module definition, after all the input/output declarations, but before any instance names:

initial $sdf_annotate("syn_sw_adder.sdf");

It is always a good idea to be sure that the synthesized design still works the way you expect it to. However, the timing information in this sdf is not very accurate because it does not account for layout. Thus, any final simulations you do with all timing should use the sdf we will get from APR.

APR

1. Change to the apr directory and take a look at the files there. The apr subdirectory should contain 4 files:
    a. dw_adder.tcl: This is another .tcl file which contains the script used to run encounter efficiently. Since the design of a block typically involves invoking encounter, verifying timing, fixing timing, resynthesis, re-invocation of encounter, etc it helps to have scripts ready to avoid long delays in redesign.
    b. dw_adder.conf: This file contains all the default information for your design along with the location of the verilog, lef, and tlf files. Be sure to verify that $my_toplevel is set to dw_adder for this tutorial (and your top level block in future designs). Also, you may be interested in the lef file. The lef file contains a "layout-like" description of the standard cells that will enable the APR tool to understand how to place the blocks without overlapping with other cells as well as route wires to the appropriate pins in the design
    c. dw_adder.save.io: This file contains pin placements for the adder. Of course, this file will not be available for your designs. This file is an edited and cleaned up version of what is created with edit->Pin Editor. It is not difficult to generate your own IO pin mapping file using the editor and we will demonstrate how to do so for you.
    d. dw_adder.sdc: This file contains basic timing information for your design. It will help the router understand false paths and driving cells. In this case it is very simple
2. type "encounter". This opens up the encounter gui window. Enter the commands in the dw_adder.tcl file into the command prompt step by step. Note the explanatory comments made in the file that elaborate on the function of each line. Again, here you can simply type "source dw_adder.tcl" in your encounter prompt. The fact that these tools from multiple vendors have standardized on tcl should make it much easier to get used to the tools.
3. The final design is output as a def file. Once the def has been exported you can import into icfb.

4. The tar file should have the cds.lib and .cdsinit for you in the base synth_tut directory. Open icfb in this directory.
5. Create a new library test_apr. In general always create the design library you want your design to be found in BEFORE you import DEF.
6. In the CIW click File->Import->DEF
7. An Import DEF form will pop up. Fill the form up as follows:
   Library Name: test_apr
   Cell Name :dw_adder
   View Name : layout
   DEF File name : apr/dw_adder.def
8. Open the layout view. Because DEF is supposed to provide an abstract view the editor that comes up is not the layout editor. We will need to fix some things up to prevent this. First click Tools->Layout to get into the layout editor.
9. Now click Edit->Search and a search form will show up.
10. Click Replace->viewName and enter layout. Then click Apply. Then Click Select All, finally Replace all before closing the search window. If you get a warning about a via that is ok.
11. Now click Design->properties and a properties form will show up.
12. Click Property within this if it is not already selected.
13. Click on the viewSubType label, then click on delete and finally ok.
14. Now save the layout with design->save
15. You will find that the layout view contains layout of cell and you now have a layout ready for LVS and DRC.
16. You will also have to import the verilog you generated from your apr flow. To do that , in your CIW, enter File->Import->Verilog and fill up the form
    Only 3 fields need to be filled
    Reference Libraries : vtvtlib25
    Target Libraray : test_apr
    Verilog files to import : dw_adder.apr.v
17. This will import a schematic view from the verilog file you have created from encounter (encounter sometimes modifies your schematic to add clock buffers) and allows you to run LVS.