Physical Design of Rachael SPARC

Benjamin LaCara and Matthew Guthaus
bsamson@soe.ucsc.edu, mrg@soe.ucsc.edu
Computer Engineering Department, University of California Santa Cruz

Abstract – The Very Large Scale Integration Design Automation (VLSI-DA) group at UCSC has a strong focus on physical circuit design, high-performance resonant clock synthesis and large-scale circuit optimization. The group is in need of a physical implementation of a microprocessor to use as a test vehicle for high-performance resonant clock synthesis using on-chip inductors that recycle the energy of the clock network.
This project is the first step in providing the group with a microprocessor which is fully synthesizable in multiple technology libraries. The physical implementation is of a 32-bit, open-source microprocessor, called Rachael SPARC. This work is branching off from the Micro Architecture Lab at Santa Cruz's (MASC) progress on developing and modifying Rachael. Our physical implementation is created using a commercial ASIC design flow with Synopsys Design Compiler and Cadence First Encounter. By interfacing to these tools through the Design Exchange Format (DEF), our resonant clock synthesis tool will add resonant clocks to the prototype microprocessor.

I. Introduction

The ASIC design flow starts with a design written in a hardware description language (HDL), such as Verilog or VHDL, and moves it through Synthesis, Floorplanning, Place & Route and Verification of Circuit. This flow also typically includes behavior simulations and static timing analysis at multiple points.

The focus of my work this summer was on Synthesis with Design Compiler and Place & Route with Encounter. The HDL was provided by the open-source Rachael SPARC microprocessor and MASC. One of the major reasons our group decided to take up the Rachael SPARC microprocessor is because it is written in Verilog. This is significant due to Verilog's popularity in the ASIC industry and in academic institutions.

The greatest objective of my work was to modify the scripts these programs used to the point where they would compile without warnings or errors while generating net-lists and layouts. This task required communication and joint work between the VLSI-DA and MASC groups.

II. Synopsys Design Compiler (DC)

For this design, we fed DC a script written in tcl which housed the necessary commands to generate a net-list with an emphasis on timing and the critical path. The basic synthesis flow of DC moves through these high-level phases: Specify libraries, read design, define design environment, set design constraints, compile strategy, optimize design, analyze and resolve design problems, and save the design database.

When specifying the libraries we designate the directory of the files which define the technology and symbols used in the design. At this stage of the process, we are using 90nm technology.

When reading in the design, the analyze command was used on all of the Verilog files. Here, the SYN_ASIC flag is set high so non-synthesizable code is ignored. Similarly, a USE_STM90 flag is set to select the technology size in each file.

The design environment and constraints are established in a script named dc_setup written by the MASC group. Here, clock periods and uncertainties, as well as input and output delays are defined. Output load constraints are also established.

For this design, we use a bottom up compiling strategy. This is done to reduce the number of unresolved reference warnings.

Given the tight timing constraints of the design, the compile_ultra command is chosen as our optimization process to receive better quality of results. This is established in the dc_setup script.

Another script written by MASC, dc_stats, is used to produce reports on area, timing, power, and quality of results. These are all attained using the "report_" command.

Following the reports generation, a Verilog net-list is produced which is used in Encounter. The command used is, simply, write.

Topographical mode in DC was experimented with for a while but ultimately proved unfruitful. This tool is used to accurately predict post-layout timing, area, and power. The mode was given up because topographical mode has unsupported commands that cause error messages to be issued and causes the script to stop executing. In Encounter there are commands which do very similar tasks that can be implemented when the flow is further developed.

III. Cadence First Encounter

The approach used for Encounter is very similar to that used with DC. A tcl script which houses all of the commands used to build the design is ran in the encounter terminal. The high-level synthesis flow of encounter moves through these phases: loading deign, floorplanning, IO and cell placement, special net routing, clock tree synthesis (CTS), in-place optimization and global and detailed routing.

First of all, given the libraries and tools available which are required for the design, Encounter has to be used on a 64 bit machine. The line used to launch Encounter follows– "encounter -64 –init ../../soc_rachael_rnode.tcl". Following this, the technology libraries and search paths are specified.

A configure file is used when loading the design. For Encounter multiple lef files are used. Along with these, a timing configuration file is used and delay and buffer footprints are specified.

The floorplanning power ring section of the script I did not modify. The floorplan specifies the aspect ratio, row utilization and space around the core.

The script originally had a command, "runN2NOpt – effort high," being used before placing the design. This command restructures and remaps elements on the critical path of the design. This command was consistently causing the script to quit prematurely, laying out nothing. This command failed with an error saying Encounter could not find the buffer or inverter libraries. Given that runN2NOpt is not required for a laid out design, the command was commented out of the code.

Placing the design is simply done with the "placedesign" command.

Following this is pre-CTS. This is as far as I got with positive results during the program this summer. Command, "buildTimingGraph," is used for timing analysis before CTS. Command, "defOut –floorplan –netlist …," is used here to create a def file which will be used for debugging at this point.

IV. Results

Over the course of this project the focus changed from optimizing timing and changing the technology libraries to having a series of scripts which would produce a functioning design. Given this, some of the resulting statistics are bad or non-existent but should be mentioned here to show what still needs to be done.

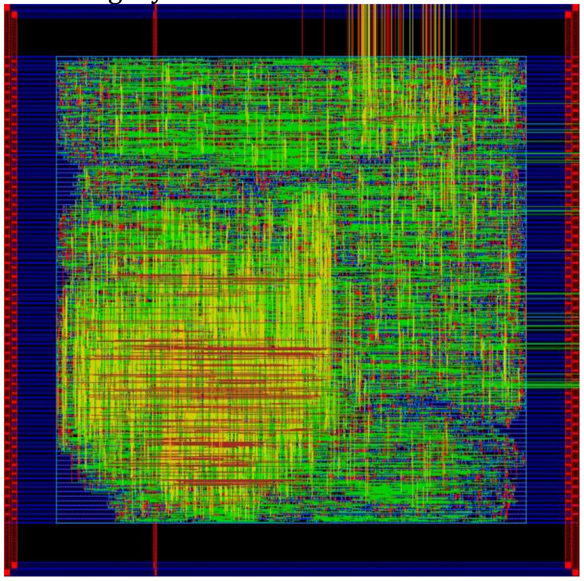
Results from Synopsys Design Compiler:

- Critical Path Slack: -30.00ns
- Critical Path Length: 0.12ns
- Critical Path Clock Period: 1.00ns
- Total cells: 13437
  - Hierarchical: 114
  - Leaf: 13323
    - Sequential Latch: 57
    - Sequential Flop: 3149
    - Combinational: 10117

Of these, the most notable problems are the negative slack and the number of latches.

If anything, the slack needs to be very close to zero if it is negative. A small timing difference from zero can often be made up in actual physical layout. Also, there should be no latches at all.

Resulting layout from Cadence Encounter:



The above layout shows the state of the design at clock tree synthesis. As the flow continues to be corrected filler cells and pins will be added. These will change the size of the design as well as the way it appears.

V. Acknowledgements

I would like to thank the National Science Foundation for providing the funding which enabled me to participate in this program. I would also like to thank the University of California Santa Cruz, SURF-IT and its staff for the opportunity to take part in this experience this summer. Finally, I would like to thank Matthew Guthaus for the opportunity to work on this project and for his guidance throughout the process.