EECS 151/251A ASIC Lab 6: SRAM Integration and Post-PAR Simulation

Prof. Borivoje Nikolic and Prof. Sophia Shao
TAs: Cem Yalcin, Rebekah Zhao, Ryan Kaveh, Vighnesh Iyer

Overview

In this lab, we will go over two very important concepts. Firstly we will look at the basics of using circuits other than standard cells in VLSI designs. The most common example of this is SRAM, which is a dense addressable memory block used in most VLSI designs. The process for adding other custom, analog, or mixed signal circuits will be similar to what we use for SRAMs.

The second concept is the post-PAR simulation with timing annotations. In the last lab, we have performed a post-PAR simulation with no delays, just to verify that functionality was not changed. In this lab, we will also simulate the delays introduced by the standard cells as well as the interconnects between standard cells.

To begin this lab, get the project files and set up your environment by typing the following commands

```
git clone /home/ff/eecs151/labs/lab6.git
cd lab6
export HAMMER_HOME=$PWD/hammer
source $HAMMER_HOME/sourceme.sh
```

If you have not done so already you should add the following line to your bashrc file (in your home folder) so that every time you open a new terminal you have the paths for the tools setup properly.

```
source /home/ff/eecs151/tutorials/eecs151.bashrc
```

You will have to modify `design.yml`, `design_gl.yml` and `design_par.yml` files and replace `<YOUR_LAB_ROOT_DIRECTORY>` with the absolute path to your lab clone's root directory.

As a final note, please make sure you run `make clean` in all your previous lab directories to minimize the space they occupy on the disk.

Differences in Files Compared to Last Lab

Inside the `src` folder, you will find the solution to last week's lab with some additions. They are very similar to what we used for the previous lab, but there is one key important difference in the coprocessor:

```
SRAM2RW32x16 sram (
    .CE1(clk),
    .CE2(clk),
    .WEB1(web1),
    .WEB2(web2),
    .OEB1(oeb1),
    .OEB2(oeb2),
    .CSB1(csb1),
    .CSB2(csb2),
    .A1(address_1),
    .A2(address_2),
    .I1(data_in_1),
    .I2(data_in_2),
    .O1(data_out_1),
    .O2(data_out_2)
);
```

This code instantiates an SRAM module, which is a dense memory unit provided by the foundry. This module specifically is a 32x16 SRAM, which means that there are 32 entries of 16 bits. This means that there is a 5 bit address for selecting those entries. In this code one port is configured to be a write port, and the other port is configured to be a read port by setting up the input parameters for the ports. The data for the write port comes from the `A_reg` signal within the `gcd_datapath` module inside the `GCD_0` instantiation of `gcd_unit` module, and the address is just constantly looping around. This sets up a history of previous values of the `A_reg` signal over time, which could be read out for debugging and verification. SRAM outputs for both ports are provided as outputs from the module, although only one of them is used in this configuration.

Because SRAMs are not made out of standard cells, and are rather built using different units that do not conform to our PAR flow, they are pre-compiled and stored in separate databases. These cells are then instantiated by Innovus as black boxes, and are connected to the rest of the circuit as specified in your Verilog. In order to generate the database that Innovus will use, type the following command:

```
make srams
```

To simulate this RTL design, a model file for the SRAM has been provided (`src/sram.v`). This file includes models for various types of SRAMs. While these models are written in synthesizable Verilog, most of the simulation models you will encounter will include non-synthesizable code. To perform the simulation, enter the following command as in the previous labs:

```
make sim-rtl
```

If you wish to see what the new outputs are doing, execute the following commands to bring up the waveform viewer and inspect the waveforms:

```
cd build/sim-rundir
dve -vpd vcdplus.vpd
```

Question 1: Understanding SRAM Models

a) Open up the verilog implementation of the SRAM. What are the different sizes available in this process?

b) Look through the code of the SRAM used in this design, understand its operating principles and function of every pin. What is the difference between `SRAM2RW*` and `SRAM1RW*` variants? You will need to use one (or more) of these SRAMs in your project.

PAR With SRAMs and Post-PAR Simulation with Timing Annotations

For small to moderate designs, the best way to make sure your circuit will work after fabrication is to perform a post-PAR simulation with delay annotations. This will account for the effects of gate and interconnect delays, while also checking for functionality. Synthesize and PAR this design:

```
make syn
make par
```

We won't be performing post-synthesis simulation in this lab. To configure the simulator for post-PAR simulation, type the following command:

```
make par-to-sim
```

You will have to type this command every time you do PAR, but once you type it you can perform as many simulations as you want so long as you only change the testbench. Now we have to perform some hacks to the flow to make our post-PAR simulation work, but while doing that we might as well take a look at how our design looks. Firstly, go to `build/par-rundir` and open up Innovus:

```
innovus -common_ui
```

When the GUI pops up, go to `File -> Restore Design`. Select Innovus as the data type, and click on the folder icon to select the database to be loaded. The last version of the design that is saved is called `post_write_regs`, so select that and press Choose and then OK. It will take a short while to load the design. Once it loads, press the `Physical View` button on the top right of the GUI, above the layer palette. Also turn off M9, V8, M8, V7 as before. You can now see the arbited, FIFO interfaced GCD coprocessor with SRAM-based debugging capability implemented in a 7 nm technology in its full glory. Note the coordinates of the bottom left corner of the SRAM.

Question 2: Understanding Floorplan Constraints

a) In which file do we specify the coordinates of the SRAM? Change the coordinates such that SRAM is in the bottom of the design instead of the top. **Note: You can only set the y coordinate to integer multiples of 1.08**

b) Change the GCD coprocessor code such that there are two SRAMs that now provide debug information for both units, instead of just one. You will have to create a floorplan constraint for the new SRAM, using the provided one as reference. Because two SRAMs won't fit the given floorplan, you will also have to modify the floorplan size to make the canvas bigger.

Now to perform the hacks. Keeping the Innovus window open, open up `par.tcl` file inside `build/par-rundir`. Find the `write_netlist` command that generates the `gcd_coprocessor.sim.v` file. Delete the `-flat` switch from this command. We are doing this to generate a hierarchical Verilog file instead of a "flattened" version where all hierarchies are removed. There are various reasons people use one over the other, but now, we want a hierarchical netlist because our delay annotation file (`.sdf` file) is hierarchical. After removing the switch, copy and paste the command to innovus shell.

The second hack we will perform has to do with the output of the `par-to-sim` target. Open up `build/par-sim-input.json`. This is a database that is generated for the simulator and includes various information about our design. Normally, you wouldn't edit these files, but for this lab, this file requires a small fix. Find the following line:

```
"sim.inputs.input_files": "<YOUR_LAB_ROOT_DIRECTORY>/build/par-rundir/gcd_coprocessor.sim.v",
```

And change it to:

```
"sim.inputs.input_files": ["<YOUR_LAB_ROOT_DIRECTORY>/build/par-rundir/gcd_coprocessor.sim.v"],
```

Save and close the file. Now return back to the root directory and execute the following command:

```
make sim-par
```

After the simulation is done, open up the waveform viewer as described previously in this lab spec. Zoom in to the signals and see that they no longer all change simultaneously at the clock edge, but rather have their own unique delays.