EECS 151/251A ASIC Lab 5: Clock Tree Synthesis (CTS) and Routing

Written by Nathan Narevsky (2014, 2017) and Brian Zimmer (2014)
Modified by John Wright (2015, 2016) and Arya Reais-Parsi (2019)

Overview

To begin this lab, get the project files by typing the following command

```
git clone /home/ff/eecs151/labs/lab5
cd lab5
```

Like last week, this lab has two parts. For the first part, we will continue to develop our GCD coprocessor by improving its performance. After that, we will continue the physical design flow by performing clock tree synthesis (CTS) and routing.

Design

One way we can improve the performance of our GCD coprocessor is by parallelizing the compute. We can do this by including multiple GCD units in our design, and routing traffic to them as they become available.

You will find that the solution to last week's lab (`fifo.v` and `gcd_coprocessor.v`) is included. You should run `make run` in `vcs-sim-rtl-coprocessor` to observe that the functional tests pass. However, an additional test has been added that checks the total number of cycles taken by the coprocessor to complete the tests. Take note of the number of cycles that the tests take without modification, as you will need it to calculate your speedup.

Your task is to edit `gcd_coprocessor.v` to improve the perfomance below 250 cycles. We will do this by using two instances of GCD.

You will find RTL that connects the datapath and controller into one module in `gcd_unit.v`. You may find this useful when refactoring the `gcd_coprocessor`, since you will need fewer wires to place both GCD instances.

You will also find stub code for an arbiter, which you should complete. We will use the arbiter to route traffic to GCD units and preserve the response ordering. Most of your design can be implemented with combinational logic, but you will need some state to remember which GCD block contains the earliest data to preserve ordering.

Question 1: Design

a) Submit your code (`gcd_coprocessor.v` and `gcd_arbiter.v`) with your lab assignment.

b) How many cycles did your simulation take? What was the % speedup?

Placement

To get started on CTS and routing, we will continue where we left off last lab. First, we will re-run synthesis and placement as shown below. To run synthesis:

```
cd dc-syn
make
```

Now we are going to run through the placement steps that we did last lab in an automated way:

```
cd ../icc-par
make init_design_icc
make place_opt_icc
```

Where the first command creates the floorplan, and the second command will place the standard cells. Open `Makefile` to see what these commands do.

You can see that the `init_design_icc` target executes IC Compiler with the following command:

```
init_design_icc_tcl     := icc_scripts/init_design_icc.tcl
...
$(init_design_icc): $(iccdp_timestamp)
...
  $(icc_exec) -f $(notdir $(init_design_icc_tcl)) \
  | tee -i $(log_dir)/$(notdir $(init_design_icc)).log; \
```

So when you run `make init_design_icc`, the script `icc_scripts/init_design_icc.tcl` is run.

Also notice the dependency for this target, `$(iccdp_timestamp)`. This target creates the `build-iccdp` directory where IC Compiler runs and copies in the relevant files, then creates the `current-iccdp` symbolic link. Another common target is `$(icc_timestamp)`, which creates the `build-icc` directory and the `current-icc` symbolic link. There are two directories created for IC Compiler: `current-iccdp` (where dp stands for design planning) for floorplan exploration, and `current-icc` where most of the flow is run.

Question 2: Makefile structure

By reading through `icc-par/Makefile`, describe in order targets are executed if you simply type `make` in `icc-par`. For each target, report whether or not IC Compiler is run (and if IC Compiler runs, what is the filename of the script it sources).

Up until now, the scripts automatically synthesized your design, created the floorplan (pads and power straps), and placed the standard cells. To view your design,

```
cd current-icc
./start_gui
```

Ignore the error message (this command expect to open the final design in the flow), then go to File — Open Design. Each step in the flow gets saved as a different cell, so you can revert to earlier steps for debugging. If you sort by the modification date, you can see the flow step ordering (which is also defined in the Makefile). Open `place_opt_icc`.

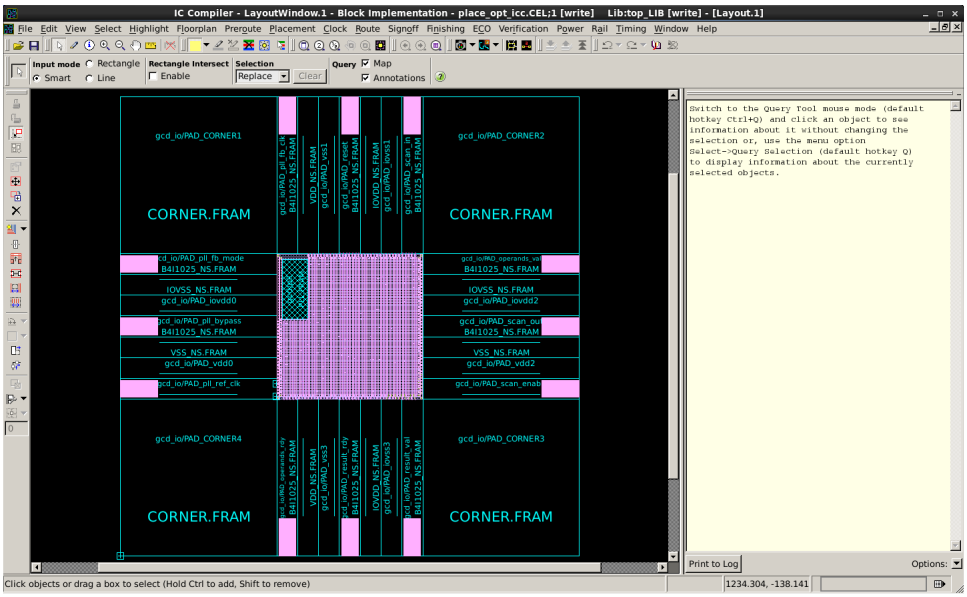

Figure 1: After placement

Click on the "Layers" tab on the left side of the screen. Hide the power straps by clicking on "M8", "VIA8", and "M9" and clicking "Apply." Zoom in to look at a cell. At this point, there is just a blackbox for each cell. On the left side of the screen, increase the level to 99 and click "Apply" again. Now you can see inside each of the standard cells. Find a flip-flop (named DFF*), and look for the pin indicators, as shown in Figure 2. Now click on the "Objects" tab, and make "Pins" and "Pin Shape" by visible (Vis) and selectable (Sel) and click "Apply" again. By mousing over the pins, you can see the layer the pins expect a connection in, and by clicking on them, you can see the shape of metal that the routing tool will connect to. Right click on the Q pin of the flip flop, and select Highlight — Net Flylines of Selected objects to see what signals use the bit stored in the flip-flop.

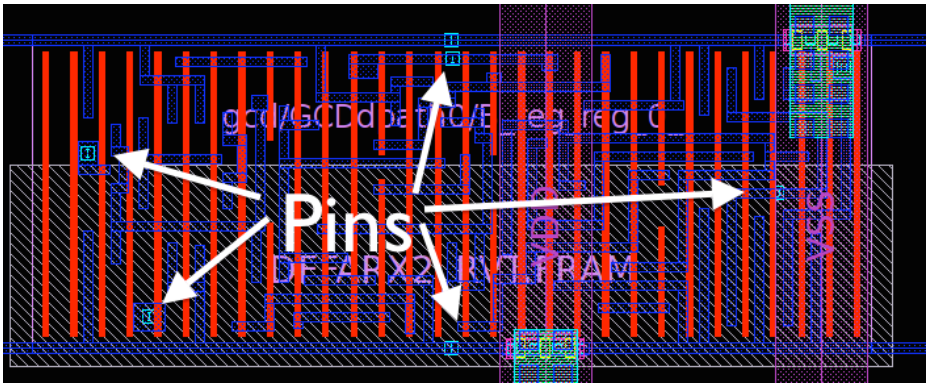

Figure 2: Pins in a flip-flop

You can also find specific cells in the design graphically. Go to Select — By Name Toolbar, searching for `*A_reg*` (wildcards are important unless you type the entire hierarchical path), and choose a cell based on the search results. Click Keep to use only the cell you've selected, and click Change Selection to select this cell. To zoom to the selected cell, go to View — Zoom — Zoom Fit Selection.

Question 3: Analyzing Placement Results

a) Submit a screenshot of the flylines of the clock input to the D flip-flops `B_reg_reg_14_` (in a similar method to highlighting the output of the Q pin).

b) What cell drives the clock input to `B_reg_reg_14_` clock pin? Submit a screenshot of the flylines of the clock input to this cell as well.

c) Run report_timing with and without wire parasitic contributions (`set_zero_interconnect_delay_mode true` and `set_zero_interconnect_delay_mode false`), and submit the critical path in the fast_clk domain for both. How much did wire delay increase the critical path by?

Now report the timing after placement for a specific path, and save this result for later.

```
report_timing -capacitance -nets -physical -path full_clock -significant_digits 3 \
 -from gcd/GCDdpath0/A_reg_reg_1_/QN \
 -to gcd/GCDdpath0/A_reg_reg_8_/D
```

Clock Tree Synthesis (CTS)

Up until this point, the clock tree has been treated as ideal—the clock arrives to every flip-flop at exactly the same time. In fact, there is now a single inverter driving hundreds of gates (and in a bigger design, thousands). If physical location wasn't a problem, clock tree synthesis would only need to add a chain of buffers with increasingly large size (or actually, more parallel buffers) like the ubiquitous "optimal logical effort" problem. But in reality, the cells will be physically far from each other and many wires will need to be added to the clock tree. Also, there are clock gating cells which need a clock input before non-gated cells. IC Compiler handles all of this for you to create a clock tree that has minimum skew and insertion delay, where skew is the difference in arrival time between the shortest path in the clock tree and the longest path in the clock tree, and insertion delay the propagation time between the source (root) of the clock tree and the flip-flops.

Now we are going to go through the steps required to create the clock tree. One useful way to debug the scripts is to place the `return` command inside a script, which allows you to interactively debug during one of the steps.

Open `icc_scripts/clock_opt_cts_icc.tcl`, and on line 30, add `return`. Then run:

```
cd ..
make clock_opt_cts_icc
gui_start
```

(If the Makefile reports this step is already completed, you can run `rm current-icc/clock_opt_cts_icc` to remove the timestamp.)

Look at the existing clock tree by going to Window — New Interactive CTS Window. Right click on `fast_clk`, then choose `Levelized Abstract Clock Graph from Selected`. Check the Show and Expand boxes next to Gate and ICG. You can see there are two clock gates (icg) that drive all of the flip-flops in the design.

Now we are going to create the clock tree. To do this use the following commands:

```
clock_opt -only_cts -no_clock_route
```

The `clock_opt` command will perform the clock tree synthesis, as well as other things such as optimization and hold time violation fixing. With the `-only_cts` flag only clock tree synthesis, clock tree optimization and clock tree routing are performed. The `-no_clock_route` flag stops the actual routing from happening so that we can see what the clock tree will look like before the routing steps. After running these commands you should be able to view what it is going to do for the clock tree. Go back to the main window and click Clock — Color by Clock Trees. Click Reload and then OK. Your view should look like below:

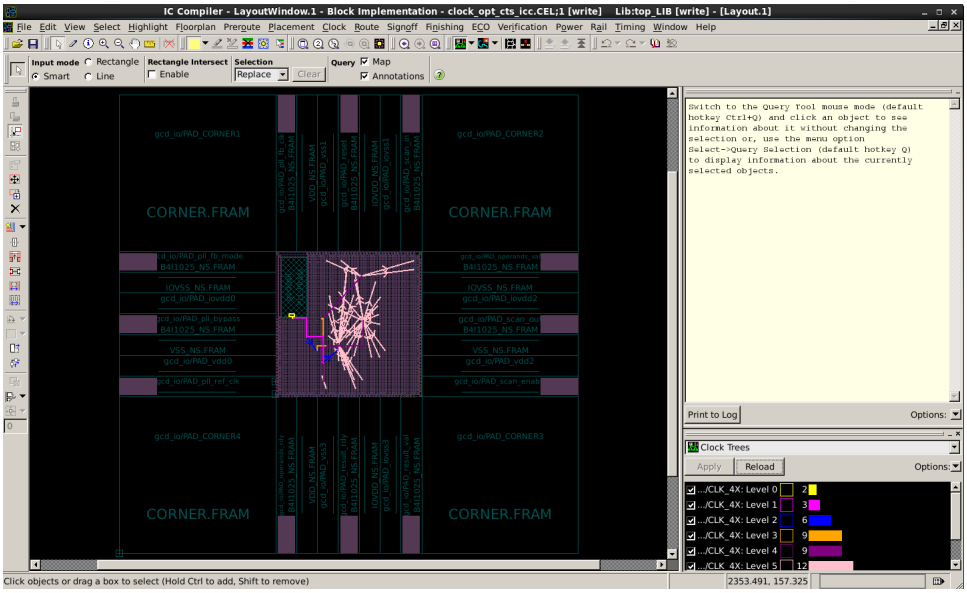

Figure 3: Clock Tree

The colors refer to different levels within the clock tree, where a level is determined by how many buffers the clock signal goes through before it gets to that point.

Now look at the clock tree timing by going to Window — New Interactive CTS Window. Right click on `fast_clk`, then choose `Latency Abstract Clock Graph from Selected`. The x-axis shows the arrival time at various elements. You can see buffers were added into the clock tree as expected, and the clock arrives almost simultaneously (becuase it is such as small design). By selecting various elements, you can see them also get selected in the design window. Notice that no routes have been created. There is no point in routing the clock tree if there are issues at this stage, because routing can only make the results worse. This is why we used the `no_clock_route` flag—it is easier to debug step-by-step if there are problems. This information is also contained in the clock tree report, which reports the skew, shortest path, longest path, and other important information.

```
report_clock_tree -clock_tree [get_clocks fast_clk]
```

Now rerun timing for the same path from the placement step:

```
report_timing -capacitance -nets -physical -path full_clock -significant_digits 3 \
 -from gcd/GCDdpath0/A_reg_reg_1_/QN  \
 -to gcd/GCDdpath0/A_reg_reg_8_/D
```

Notice that the clock network is now included in the timing report.

Question 4: Timing after CTS

Did the insertion of the clock tree help or hurt the path `-from gcd/GCDdpath0/A_reg_reg_1_/QN -to gcd/GCDdpath0/A_reg_reg_8_/D`? By how much? Why?

One great tool to understand timing inside your design is to go to Windows — Timing Analysis Window. Click "Apply" and then "Ok". You should have a window like the one below:

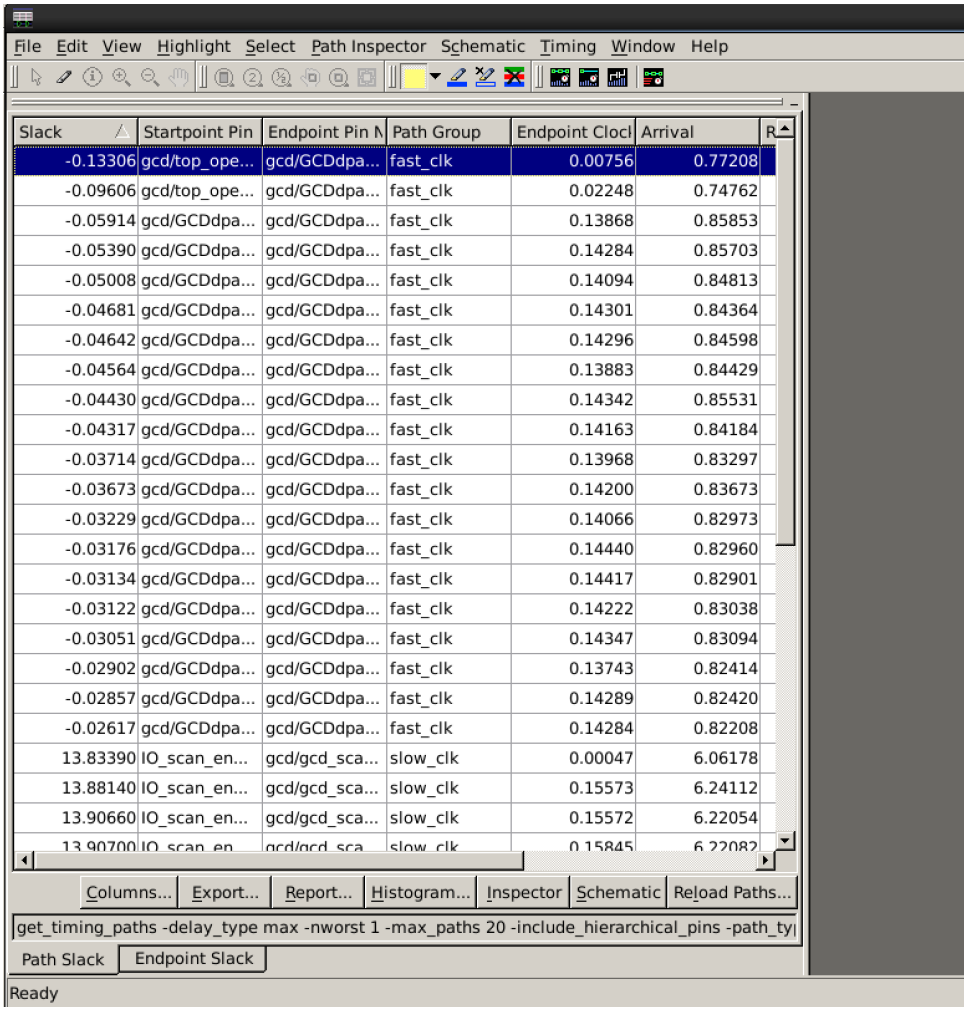

Figure 4: Timing Window

As you should be able to see, the current design does not meet timing, since there are multiple paths with negative slack. There are a lot of interesting ways to visualize this information inside of this window, so figure out how to show a histogram of the path slack at this point. In the histogram, you should be able to click on one of the bars and it will display the paths that make up that bin.

Next we need to connect the new cells we added to the design to the power grid. Until now, every cell in the design came from the synthesized netlist, but now that IC Compiler has physical information, it will need to modify the design by changing gates, inserting gates, and resizing gates, and this will change the netlist.

```
derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS -ground_pin VSS
```

Because there is a clock tree, now there is a possibility for hold times.

```
remove_ideal_network [all_fanout -flat -clock_tree]
set_fix_hold [all_clocks]
```

These commands above delete the ideal network from the clock tree, and also let the tool know that it needs to take that delay into account. The second command tells the tool to fix hold time violations on all of the clock paths. The last line saves the design as a different name than previous so that we can come back to this at any point later. Now open the Timing Analysis Window again, and report the critical path.

Skew in the clock tree can actually be useful, by extending the clock period on critical paths. Now that the clock tree has been inserted, let IC Compiler try to optimize timing again.

```
extract_rc
clock_opt -no_clock_route -only_psyn -power
derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS -ground_pin VSS
```

The first line tells the tool to extract the wires in the design for their parasitic resistance and capacitance, and the second line tells the tool to perform the optimization for the clock tree network. With the `-no_clock_route` and `-only_psyn` options nothing will get routed, it will just get optimized. This command will probably cause your circuit to meet timing again! Last, there are actually no wires connecting the clock pins together. We will use a different command for the routing of the clock nets:

```
route_zrt_group -all_clock_nets -reuse_existing_global_route true
```

The `route_zrt_group` command is the command that we will be using to do the actual routing. We route only the clock nets (`-all_clock_nets`), because these are much more critical than regular signal routes.

Question 5: Clock Tree

Print a screenshot of the highlighted longest and shortest path in the clock tree (fast clock, not slow clock). Hint: For any of the interactive windows (either CTS or Timing), selecting paths will highlight them in the design window.

Question 6: Changing the Design

Clock routes are made with different metal rules, because resistance of wires can contribute to clock skew. Experiment with different non-default routing rules that change available clock cells and routing options (common_cts_settings_icc.tcl) and report the results.

Don't bother saving the design, just close it. Now use the built in scripts to run all of the clock tree synthesis steps. You should remove the `return` you added to line 30 of `icc_scripts/clock_opt_cts_icc.tcl`.

```
rm current-icc/clock_opt_cts_icc
make clock_opt_route_icc
```

Routing

So far only the clock cells have been routed, but we need to route every connection in the design. Global routes are top level wires that span large distances on the chip, so often you want to route those first to make sure that they can actually connect properly. After those are in place, then you can route the other signals around the global route (called detail route).

Add `return` to line 25 of `icc-scripts/route_icc.tcl`, and run:

```
make route_icc
gui_start
```

```
report_preferred_routing_direction
```

The above command reports what direction the different metal layers are supposed to be routed in. The output can be seen below:

```
****************************************
Report : Layers
Design : top
Version: G-2012.06-ICC-SP3
Date   : Wed Oct  1 10:02:04 2014
****************************************

Layer Name                    Library             Design              Tool understands
M1                            Horizontal          Horizontal          Horizontal
```

```
M2                              Vertical            Vertical            Vertical
M3                              Horizontal          Horizontal          Horizontal
M4                              Vertical            Vertical            Vertical
M5                              Horizontal          Horizontal          Horizontal
M6                              Vertical            Vertical            Vertical
M7                              Horizontal          Horizontal          Horizontal
M8                              Vertical            Vertical            Vertical
M9                              Horizontal          Horizontal          Horizontal
MRDL                            Vertical            Vertical            Vertical

1
```

By default the routing directions should be all the same for the library, design, and what the tool understands. You can change these directions on a per-layer basis, but unless there is a good reason to do so the defaults are usually good options. If you do change any of the layers, make sure that whatever you are trying to do is still routable - for example, do not create power straps below the top layer grid horizontally so that they cannot connect down to the standard cells.

The following command will show what technology files the tool is using for parasitic resistance and capacitance information.

```
report_tlu_plus_files
```

And the output should appear like this:

```
****************************************
Report : tlu_plus_files
Design : top
Version: G-2012.06-ICC-SP3
Date   : Wed Oct  1 10:03:43 2014
****************************************

  Max TLU+ file: /home/ff/eecs151/labs/stdcells/synopsys-32nm/multi_vt/tluplus/max.tluplus
  Min TLU+ file: /home/ff/eecs151/labs/stdcells/synopsys-32nm/multi_vt/tluplus/min.tluplus
  Tech2ITF mapping file: /home/ff/eecs151/labs/stdcells/synopsys-32nm/multi_vt/techfile/tech2it

1
```

This tells us where the tluplus files live, which contain the information for the wiring parasitics. Open up the max tluplus file and look at the top. A tluplus file is generated from an ITF file, which stands for Interconnect Technology Format. For each metal layer and dielectric combination there are a few parameters that need to be set. Let's take a look at one set of these:

```
#     CONDUCTOR M9 { THICKNESS = 0.22 WMIN=0.16 SMIN=0.16 RPSQ=0.28 }
#     DIELECTRIC D9 { THICKNESS = 0.5 ER=3.9 }
```

This creates a M9 metal layer with a D9 dielectric layer, with a few other parameters specified. The `THICKNESS` is the thickness of the metal, the `WMIN` is the minimum width for that metal layer, the `SMIN` is the minumum spacing for the layer, and the `RPSQ` is the resistance per square of the metal layer. For those of you unfamiliar with resistance per square, this number is taken and multiplied the length of the wire and divided by the width to get the actual resistance. For the dielectric the `ER` is the relative permittivity of the dielectric material.

Continuing with the script, enter the following commands:

```
route_opt -initial_route_only
save_mw_cel -as $ICC_ROUTE_CEL
```

The `route_opt` command will route the design as well as perform postroute optimization at the same time. The `-initial_route_only` flag makes it only do the initial routing stage, and will not actually perform the optimization. We do this first, and save it so we can have a checkpoint to come back to.

Issue the following commands:

```
route_opt -skip_initial_route -effort low -power
derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS -ground_pin VSS
```

This time we skip the initial route since it has already been done, and set the effort to low so that it runs through faster. There are other options for the `route_opt` command to try to improve the performance of the routing. However, this design is not large enough to fully exercise these other options and therefore we will not be discussing them. More information can be found in the man page for `route_opt`.

If the design does not meet timing now do not worry, that will be fixed in the signoff stages later in the design process, which will basically boil down to running the core at a slower frequency. The tools can only optimize so much, and once they hit their limit there is not much else you can do without a bunch of manual intervention.

Question 7: Reporting Options

a) Using what we have learned in this lab and previous labs, report the critical path. Show the command and the result. Does the design meet timing? If not, just report that it doesn't, no need to try and fix it for now

b) Report how much power IC Compiler says the design is going to dissipate. Show the command and the result. Please note that IC Compiler has no notion of actual switching activity so this may not be the most accurate power number.

c) Report the fanout of the nets. Show the command to use, and the signal with the maximum fanout and maximum capacitance (these may not be the same net, but you should only need one command to get both).