UCLA EE 201A -- VLSI Design Automation – Winter 2024
Course Project: Chip Stitch Boundary search
TA: George Karfakis

******* IMPORTANT *******

- Run your experiments and do your work on SEASnet server `eeapps.seas.ucla.edu`
- You will work in teams of two students.
  - How you coordinate your work is up to you. We recommend a version control system like Git.
  - Be careful not to reveal your project directory to other teams!
- You can get the latest version of the materials at `/w/class.1/ee/ee201o/ee201ot2/2024_labs/project/`
- **Due Dates:**
  - **1) Midterm Progress Report – Midnight, Tuesday 27th of Feb**
  - **2) Final Code Submission – Midnight, Thursday 14th of March**
- Start early! This project will be challenging. There are also expected license and server load issues from this as well as other courses in the last few weeks. You should try and get most work done before that.
- See the end for important submission instructions.

Project Overview

Modern chips are fabricated using a photolithography process. A glass mask is first patterned with the design layout and then lights shines through it to expose a photosensitive resist in the wafer to pattern the wafer. The mask (or reticle as it is commonly known as) has maximum size of 26mm x 33mm or 858mm^2. This is why you never see any common chip larger than roughly this size. If one wants to build chips larger than this reticle size limit (e.g., large image sensors), one has to "stitch" two mask exposures on wafer (i.e., load and expose one mask on the lithography tool; unload and load and expose the second mask). This act of stitching can have errors (e.g., two masks once loaded are slightly misaligned). This can cause any design shapes crossing the "stitching boundary" to have low yield. As a result, either those shapes have to be widened in size or one has to avoid having shapes on stitch boundary.

A variant of this design stitching problem has emerged in context of extreme ultraviolet (or EUV) lithography. EUVL is used in technologies 7nm and below. Unlike previous lithography schemes, the wavelength of light used is 13.5 nm as opposed to 193nm. This allow an improvement in resolution and therefore scaling of technology nodes. But even it can run out of steam at ~2nm or so. Therefore, the semiconductor industry is banking on another improvement called "high NA EUV". Without going into the nitty gritty details of it (which I sometimes cover in ECE201D), this patterning scheme forces the effective reticle size to shrink by 2X in one dimension (i.e., maximum design size becomes roughly 26mm x 16mm). So again for large SoCs, stitching is required for all layers of the design that use high NA EUV.

The goal of this project is to minimize the layers of the design that need complex stitching. For this to happen, ***no*** design shapes should cross the stitch boundary on that layer. Assume the stitch boundary to

be 1µm wide by default. Then also no design shapes should exist in this stitch boundary band which is 1µm wide and spans the length of the chip. For the front-end (transistor) layers, you can accomplish this by not placing any cells in this stitch band. For wiring layers, you must not have routes crossing this band on the designated "no stitch" layers. This can usually be accomplished by putting placement or routing blockages in the P&R tool like Innovus. Of course, this wont come for free. Putting these restrictions can worsen delay or power of the chip.

Your goal in the project is two fold: (1) identify the right stitch boundary location in the 40%-60% width of the design region. (2) for that stitch boundary, find the maximum number of layers that can be "no stitch". Your figure of merit will depend on change in chip delay/power due to stitching and how many layers are "no stitch". The more, the better. Exact FOM will be announced later.

The midterm checkpoint would be for you to show that you have a script that can make Innovus honor a stitch boundary on designated layers for a benchmark design.

Final project report will build up on this to develop and submit the code which optimizes the stich boundary and the FOM automatically for any given design.

Notes:

- *You will find it useful to refer to the OA tutorial from class and the official OA documentation.*
- *It is recommended that you adopt version control such as Git for collaborating with your teammate. If you are not comfortable with Git, you can read Mark's tutorial at https://markgottscho.wordpress.com/2015/01/24/a-brief-git-tutorial-for-collaborative-research/ . We also recommend beginners start with www.gitimmersion.com .*
- *Please use Piazza for asking (and answering) questions before coming to office hours.*
- *Since you are expected to work on the assignment in teams of two, do not provide solutions or compare results with other teams.*

Checkers

We have included two checkers to make the design process easier. You can call them by simply calling ./run_combined_checker.csh $args or

The **combined checker** takes in 2 arguments: Your post-routing verilog file and optionally your blockage_definitions.yaml file.

Then, assuming these files define a design that has already been placed & routed, it loads it into innovus and extracts:

DRC violations (any of them indicate your design is faulty)

Setup/Hold slack and if it is negative

Core and Cell area

Wirelength(s) per metal layer

You can use the performance checker to ensure that your added blockages have not somehow broken the design, and that the design files you have are still physically realistic.

If provided with the blockage_definitions.yaml file, it will also perform a place and route blockage check to ensure innovus has honored your blockages.

Midterm Status Report

For the midterm checkpoint, we would like you to show that you have a script that can make Innovus honor a stitch boundary (place + route) for the designated benchmark design (s1494). You should aim for a 1µm boundary width in roughly the middle of the design.

The benchmark design has 4 metal layers, so we would like you to run 7 tests:

- P&R without any blockages
- P&R with a place blockage
- P&R with place blockage, and route blockage on M1
- P&R with place blockage, and route blockage on M1,M2
- P&R with place blockage, and route blockage on M1,M2,M3
- P&R with place blockage, and route blockage on M1,M2,M4
- P&R with place blockage, and route blockage on M1,M2,M3,M4 (!)

This should be quite simple to implement. You will need to pay close attention to both Innovus manuals provided to figure out which instructions to use.

Some of the tests above may not terminate, or may produce errors. For the midterm report, this is fine. However, you need to explain why that may happen.

Please run both performance and boundary checkers for every test result. You can *and should* automate this.

Hint: You need to define separate blockages for place and separate blockages for route. Check to see the differences between *Core* and *Design* area. Your place blockage should be in the core area, and your route blockage should be in the total design area.

Deliverables:

1. Short report (no more than 1 page/slide) showcasing the commands you used to achieve the stitch boundary. In the report, you need to explain why certain tests may have resulted in DRC errors / not terminated properly. Report if any tests fail the boundary checker.
2. A plot of how performance metrics (setup slack, area and wirelength) are affected by each of the 7 tests we defined above. You can use the performance checker to extract these metrics.

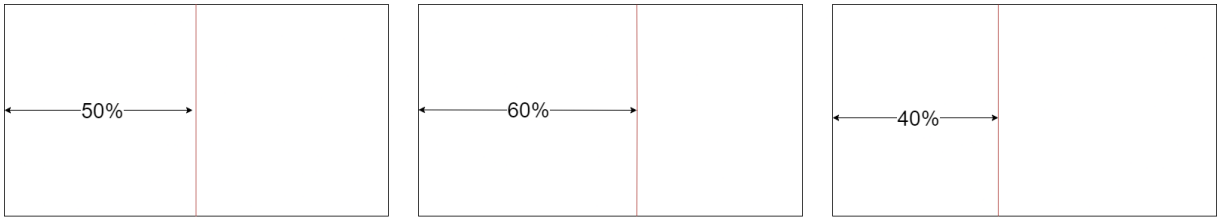
Final Deliverables

For the final submission, we would like you to create a script that, automatically for a given design, searches many different stitch boundaries across the chip in order to split it into 2 parts.

Each part can constitute up to 60% of the total design area (so 60+40 or 50+50 are both allowed).

Your script should implement some sort of search/optimization algorithm to search between different possible stitching configurations for the design, and pick the best configuration with the lowest delay and area tradeoffs.

A trivial (and ineffective) example approach would be to investigate splitting the design into just 3 different configurations as follows:



For each of these configurations, you would then gather delay/power measurements for each possible number of metal layers with the stitch boundary. You can then compute a FOM score and determine which configuration is best.

You should obviously try and find a better search space than the one listed above to traverse for this project.

You should consider many approaches, such as investigating individual layers for congestion, perhaps even using OA, to figure out where to best place your boundaries.

We want to see unconventional and out-of-the-box thinking. Your points will depend not only on the performance of your design, but also on your through process.

You would then create a score metric with arbitrary weights of your own choosing (can be based on our FOM score weights if you want), deciding which configurations give the best performance.

There are strict time constraints due to the number of students. Your script *cannot* take more than 1 hour and 30 mins to complete a run.

There will be contest points, depending on how well your design metrics compare to other students.

Figure of merit (your score on which your contest points will be measured, higher is better):

Where:
L = number of metal layers with a route blockage.

T = setup timing slack

D = number of DRC violations

WL = total wirelength

P = Successful placement blockage definition

Score = $\alpha*L + \beta*T - \gamma*D - \delta*WL + \varepsilon*P$

The exact parameters for $\alpha,\beta,\gamma,\delta,\varepsilon$ will be determined later.

Technical details for final script

We have decided that the blind test case will be a version of the FPU design with a different UTIL variable.

This means you can use the innovus_skeleton_fpu.tcl file AS-IS for the blind design.
All you have to change is the UTIL, which will be provided as the first argument into your script.

Afterwards, your script should then go into your innovus FPU tcl script, and insert the new UTIL in line 49 of the script. I have included a short example python script that does this for you as an example.
It should then add a place and route blockage into the script wherever you wish. I recommend adding it after the floorplan command in line 50.

It should run innovus using the tcl script you wrote above, extract performance information, and then decide how to continue with more simulations (with different blockage placements) to maximize performance.

At the end, it should generate an output_files.yaml file (example located in project directory), that includes the exact filepaths (relative to project root) of the generated def , verilog, lef, gds, and the blockage_definitions.yaml file.
This output_files.yaml file is what we will use for grading, so it is extremely important that this is generated successfully.

A recommendation due to the time limit is that you should keep your “best” results in your output_files.yaml and keep regenerating it (or the files) every time innovus finishes. As such, if we abruptly stop your script due to hitting the time limit, your output_files.yaml still contains your best solution.

Basic example rundown of script (please feel free to change this around if you want to try something different)

This is written in python-pseudocode.

```
new_util = arguments[1]

replace_fpu_tcl_script_util(new_util) # replaces the innovus_skeleton_fpu.tcl util with the util provided.

run_innovus_skeleton() # runs the FPU skeleton script (or some other baseline script) to simply extract design and core area information without any blockages

core_area, design_area = read_areas_from_result_file() # reads the core and design area from the results you have generated

while (time < 1.5 hours OR no_new_places_to_search):
    new_blockage_coordinates = search_algorithm() # the meat of this project, figuring out which search algorithm to use and where to place blockages.

    add_new_blockage_to_tcl(new_blockage_coordinates) # changes the innovus_skeleton_fpu.tcl script to provide the new blockages

    run_innovus_with_new_blockage_tcl()

    results = run_combined_checker()

    if results > best_results:
        save_results_to_output_files_yaml()
    continue
```

Deliverable is your script, output_files.yaml, and all other intermediate files you are using.

Project Submission

You should submit your tool's complete source code (the `src/` subdirectory only). Do not include benchmark test cases, temporary files, etc. A complete submission should thus contain:

- `src/` subdirectory
- An executable .csh script that runs your workflow. It should be named "project.csh".
    - It should take 1 argument as an input: the input UTIL for the FPU design.
    - You can assume it will be ran in the project root folder (same directory as innovus_skeleton.tcl and all other files).
- A properly formatted output_files.yaml file.
- README file with any special considerations.

Be absolutely sure that you can run the complete flow without problems. Otherwise, your submission cannot be graded!

SUBMISSION INSTRUCTIONS (Read carefully)

- All necessary code must be tarballed and submitted as a single archive file on `eeapps.seas.ucla.edu`. Presentation slides (PowerPoint or PDF format) and midterm progress report (PDF) should be posted on bruinlearn by the respective deadlines.
    - For final code submission, create a directory named as follows: `GroupID_Lastname1-Firstname1_UID1_username1_Lastname2-Firstname2_UID2_username2_Project/`. For example: `Group1_Bruin_Joe_123456789_joe_Bruin_Josie_987654321_josie_Project/`
        - Include all files mentioned in the Project Submission section.
        - Compress and archive this directory using tar/gzip to have a single submission file named `GroupID_Lastname1-Firstname1_UID1_username1_Lastname2-Firstname2_UID2_username2_Project_pinXXXX.tar.gz` (Make up a 4-digit numeric PIN of your choice to substitute for `XXXX`). For example: `Group1_Bruin_Joe_123456789_joe_Bruin_Josie_987654321_josie_Project_pin1234.tar.gz`
        - Submit tarball by copying it to `/w/class.1/ee/ee201o/ee201ot2/submission/project/`
        - IMPORTANT: Make sure all files you submit, as well as the tarball, have full read and execute permissions to group and others or we may not be able to grade your lab. Do this using `chmod -R go+rx FILE_OR_DIRECTORY`
- **Late submissions will not be accepted** (write/execute permissions to the submission directory will be removed). If you cannot finish the entire assignment on time, submit whatever you completed. **No submission = no points.** You are welcome to overwrite your submission as many times as you like before the deadline.
- Some students may worry that their work could be visible to other students. We try to reduce this chance by disabling read permissions on `/w/class.1/ee/ee201o/ee201ot2/2024_labs/project/` directory so that other students cannot list its contents. Thus, they will not know the filename and cannot open your submission unless you give them your full name, student ID number, SEAS username, and arbitrary 4-digit PIN that you substituted for `XXXX` earlier on the tarball. This also means you will not be able to list the directory contents to see if your own submission worked – you can only write to it! In short, please do not give your classmates this information or collaborate on homeworks. The PIN can be whatever 4-digit number you want -- it is just to reduce the likelihood of another student guessing your full file submission path. You can change it for every lab submission if you like.
- **Test your submission before the deadline!**