RISC-V: ASIC-Design Team

Charlie Liu
Electrical and Computer Engineering
Rice University
Houston, United States
cl228@rice.edu

Kelly Chan
Electrical and Computer Engineering
Rice University
Houston, United States
kc162@rice.edu

***Abstract*—The reduced instruction set computer five (RISC-V) ecosystem is growing rapidly, but a gap remains in transitioning register-transfer level (RTL) designs to manufacturable chips from the previous semesters of the RISC-V Capstone. This project aims to bridge that gap by implementing a complete application-specific integrated circuit physical design flow. It will begin from a basic ELEC 527: VLSI Systems Design framework, and then define a custom process to achieve the final Graphic Data System II (GDSII) file while optimizing power, performance, and area (PPA). Our solution includes adapting previous flows to our RISC-V core using industry standard tools such as Design Compiler and Innovus to create our own automated flow, beginning with synthesizing the RTL all the way to generating a GDSII layout. By next semester, our workflow will be completed using the basic substitute file and it should be adapted to the core design. Eventually, it will be taped out into a manufacturable chip.**

***Index Terms*—RISC-V, physical design, RTL, GDSII**

I. Introduction

First, it is important to understand why reduced instruction set computer five (RISC-V) is important, and how it impacts the engineering industry and markets. According to Synopsys, "RISC-V is an open-source instruction set architecture used to develop custom processors for a variety of applications, from embedded designs to supercomputers." [1] Since its inception in 2010, it has quickly and exponentially become an integral part of computing. Its skyrocketing popularity and projected market growth can be attributed to a few key features, including but not limited to:

- "Its open-standard nature, which allows collaboration and innovation across the industry." [1]
- "Common ISA, which helps make software development easier since all processors could potentially use the same architecture. Designers can use the same base ISA, from simple embedded devices to the largest supercomputers, tailoring their device to the needs of the market. Compared to previous ISAs, RISC-V ISAs have unique features and can be customized based on their requirements." [1]
- "Availability of smaller, energy-efficient, and modular options." [1]
- "Security features, which are available through open-source reference designs, software composition analysis tools, and security extensions. In addition, its open-source nature means that the entire RISC-V architecture can be scrutinized closely in the public domain, eliminating back doors and hidden channels." [1]

Thus, our project aims to take advantage of these features. The SwitchMCU, a capstone four semesters in the making, utilizes a custom RISC-V 32-bit core that has the ability to execute C programs, as well as allow for future custom instructions and additional peripherals. It is programmed as a general-purpose biomedical application processor, supports a 5-stage pipeline, and uses Harvard architecture. Furthermore, it contains synchronous and asynchronous First-In, First-Out (FIFO) data structures for data transmission, Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI), and General-Purpose Input/Output (GPIO) peripherals. More can be found on the official GitHub website. [2]

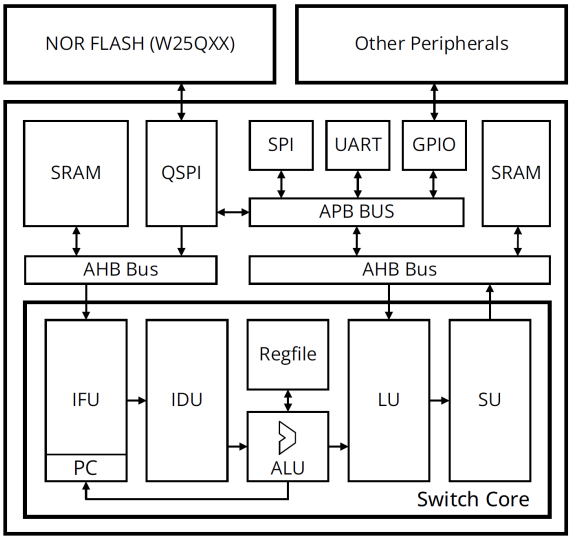

Fig. 1. System-Level Diagram of the RISC-V Based SwitchMCU.

The project can be broadly divided into two main sections: the core design and the physical design, which is the main focus of this paper. Physical design can be defined as "the process of transforming a circuit description into the physical layout, which describes the position of cells and routes for the interconnections between them." [3]

The core team designs, debugs, and verifies the RISC-V core, and will ultimately produce synthesizable RTL code to be passed along to the physical design team, which consists of the two authors. Once the Verilog code is received, it will be synthesized and ultimately taped out into a reproducible, manufactured chip.

As mentioned previously in the abstract, there is currently no standardized RISC-V Application-Specific Integrated Circuit (ASIC) workflow at Rice University. This is compounded by a more generalized problem in open-source physical design flows, which is confusing or poor documentation. This makes it difficult or even impossible for reproduction and restricts access to only those who are very familiar with the subject.

Therefore, the main assignments of this semester for the physical design team are providing fully documented instructions for each step of the physical design process, a GitHub repository (S25_pd) with all necessary TCL script files and code, and successfully synthesizing the RTL all the way to generating a GDSII layout. By next semester, the design should meet performance, power, and area (PPA) constraints, and also be ready for actual tapeout with minor modifications.

II. Methods

However, current work from the previous semesters requires revisions and additions before it is able to be transferred to a manufacturable chip. Therefore, a simple register file (`regfile.v`) from ELEC 526: High Performance Computer Architecture will be used as a substitute until the core team is able to complete the processor.

This file is then fed through a simplified physical design flow provided by ELEC 527: VLSI Systems Design Framework. These steps begin with a functional simulation in Questa. The design and testbench files, written in Verilog are compiled, simulated, and run in the GUI. The waveforms are then analyzed to check for any potential issues. Next, the output(s) are passed along to Synopsys Design Compiler for synthesis, which is the process of converting RTL into a gate-level netlist using standard cells. The design is optimized for performance, power and area (PPA) constraints. The synthesized Verilog output is then once again examined in Questa. The final component is place and route in Innovus. The software reads the netlist, places the standard cells within the chip layout, routes the interconnections between the cells, and generates both a physical design report and a GDSII layout. Since the AND structure files provided for this assignment have already been validated and refined, many intermediary steps between synthesis and place and route have been excluded.

Physical Design Flow

The full edited physical design flow for this project can be seen below in Fig.2:

1) **Synthesis** - Originally, synthesis was attempted using Library Compiler with an input of `.lib` and an output of `.db`. A TCL script was created to convert all `.lib` files to `.db` files automatically due to the high volume of these found in the SkyWater 130nm PDK. The `.db` file is forwarded to Design Compiler, which outputs a `.vg` file. (For the final design, the file will be also verified in Questa Post DC after initial synthesis with the help of the core design team.) But because of the complexities present in the modified Skywater PDK used in the project, this approach proved problematic in later stages. More about this is discussed in the later section. However, these steps are still applicable for standard synthesis using a general PDK.
2) **Floorplanning in Innovus** - The file is sent to Innovus for floorplanning, which "is the process of creating core area, specifying core to I/O boundary spacing, standard cell rows, placing I/O pins, placing macros using macro guidelines, and adding placement blockages and halo." [4]
3) **Place and Route (P&R)** - Like floorplanning, place-and-route is achieved in Innovus. This is the physical implementation stage where logic cells are arranged and interconnected on a chip, and then streamlined. Since there were numerous `.lef` files in Skywater with duplicates, a TCL script was written to do this explicitly while avoiding the repetitive cases.
4) **Clock Tree Synthesis (CTS)** - The final step in Innovus is CTS, which "aims to minimize the routing resources used by the clock signal. Additionally, it minimizes the area occupied by the clock repeaters while meeting an acceptable clock skew, a reasonable clock latency, and clock transition time." [5]
5) **Parasitic Extraction (PEX)** - The `.gds` file is then given to StarRC for parasitic extraction, also known as Resistance-Capacitance (RC) extraction. This process involves determining both the resistance and capacitance values of interconnections and devices in a circuit layout to more accurately model signal delays and timing behavior.
6) **Static Timing Analysis (STA)** - The extracted file(s) are scrutinized in PrimeTime, during which the static timing analysis "provides full chip timing with signal integrity, advanced node accuracy, and ECO guidance." [6] As the name suggests, this stage is crucial for ensuring correct and reliable timing in the design.
7) **Design Verification** - The last part of the physical design flow is design verification using Calibre, which can be dissected into two parts: Design Rule Checks (DRC) and Layout Versus Schematic (LVS). DRC is used to verify that the physical layout of the chip adheres to the restrictions and requirements set by the manufacturer. Meanwhile, the purpose of the LVS is to verify that the layout of the chip is representative of the netlist.

After all seven phases have been successfully completed, the design can be sent to the manufacturer for tapeout. Actual tapeout can span across a period of several months.

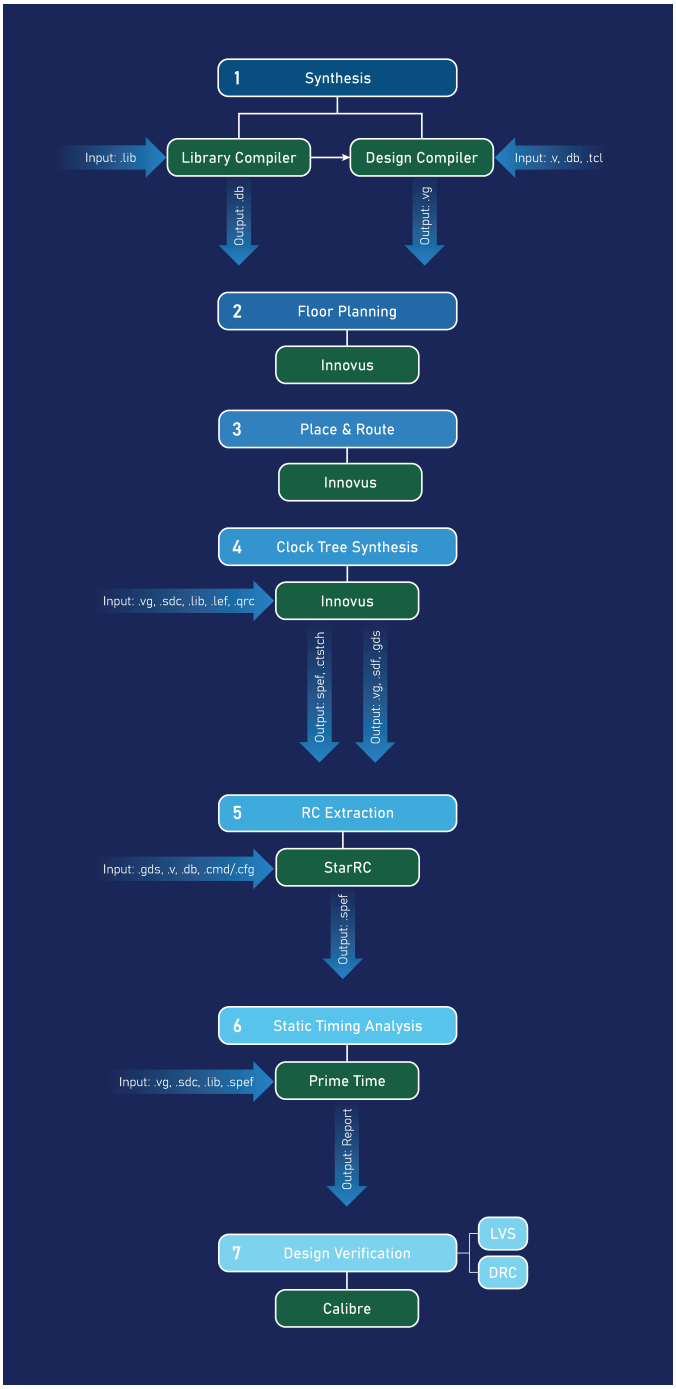

Fig. 2. Physical design flow.

III. Results

Current progress at the time of this report is around 60%. For the next semester, there remains: RC extraction, static timing analysis, physical verification and post synthesis simulation for the regfile. After successfully completing the physical design flow with this stand-in file, the core design (or a similar alternative if it has not yet been completed) will be adapted to the framework. Eventually, it will be taped out into a fabricated chip. Since the core team has not finished all the RTL code, the following results are based only on the register file block.

The total number of cells used, along with the area, timing, and power results based on the SkyWater PDK, are shown in Fig. 3:

```
Total 4664 cells                                   42176.700007
---------------------------------------------------------------
data required time                                       340.08
data arrival time                                         -3.39
---------------------------------------------------------------
slack (MET)                                              336.69

                  Internal         Switching         Leakage           Total
Power Group       Power            Power             Power             Power   (   %    ) Attrs
---------------------------------------------------------------------------------------------
io_pad            0.0000           0.0000            0.0000            0.0000  (   0.00%)
memory            0.0000           0.0000            0.0000            0.0000  (   0.00%)
black_box         0.0000           0.0000            0.0000            0.0000  (   0.00%)
clock_network     0.1038           0.0000            0.0000            0.1038  (  77.05%)  i
register          2.2105e-04       9.6671e-05        1.5350e+04        1.5668e-02  (  11.63%)
sequential        0.0000           0.0000            0.0000            0.0000  (   0.00%)
combinational     1.8742e-03       5.0947e-03        8.2911e+03        1.5260e-02  (  11.33%)
---------------------------------------------------------------------------------------------
Total             0.1059 mW        5.1914e-03 mW     2.3641e+04 nW     0.1347 mW
```

Fig. 3. Area, timing, and power after synthesis.

Fig. 4 and Fig. 5 illustrate DRC verification results and final layout after detail routing

```
innovus 72> verify_drc
#-check_same_via_cell true             # bool, default=false, user setting
 *** Starting Verify DRC (MEM: 3001.0) ***

  VERIFY DRC ..... Starting Verification
  VERIFY DRC ..... Initializing
  VERIFY DRC ..... Deleting Existing Violations
  VERIFY DRC ..... Creating Sub-Areas
  VERIFY DRC ..... Using new threading
  VERIFY DRC ..... Sub-Area: {0.000 0.000 103.040 103.040} 1 of 9
  VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {103.040 0.000 206.080 103.040} 2 of 9
  VERIFY DRC ..... Sub-Area : 2 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {206.080 0.000 309.120 103.040} 3 of 9
  VERIFY DRC ..... Sub-Area : 3 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {0.000 103.040 103.040 206.080} 4 of 9
  VERIFY DRC ..... Sub-Area : 4 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {103.040 103.040 206.080 206.080} 5 of 9
  VERIFY DRC ..... Sub-Area : 5 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {206.080 103.040 309.120 206.080} 6 of 9
  VERIFY DRC ..... Sub-Area : 6 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {0.000 206.080 103.040 306.000} 7 of 9
  VERIFY DRC ..... Sub-Area : 7 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {103.040 206.080 206.080 306.000} 8 of 9
  VERIFY DRC ..... Sub-Area : 8 complete 0 Viols.
  VERIFY DRC ..... Sub-Area: {206.080 206.080 309.120 306.000} 9 of 9
  VERIFY DRC ..... Sub-Area : 9 complete 0 Viols.

  Verification Complete : 0 Viols.

 *** End Verify DRC (CPU TIME: 0:00:01.8  ELAPSED TIME: 0:00:02.0  MEM: 256.1M) ***
```

Fig. 4. DRC verification results.

IV. Discussion

One major setback was multiple issues in the initial attempts at routing. Originally, duplicate information from the Skywater 130nm PDK was merged, creating issues that needed correcting before the physical design flow could continue. The three

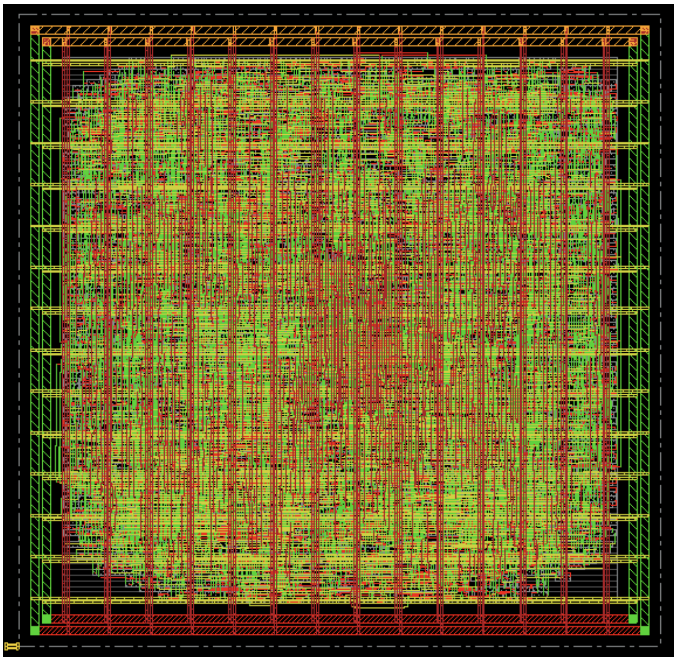

Fig. 5. Layout results from the place-and-route stage.

main barriers found were routing issues, failure to the build timing graph, and GDS file incompatibility.

Therefore, it was necessary to take a step back and repeat synthesis using a different approach. By modifying the python code provided by Timothy Edwards on Github, the .lef were successfully merged. [7]

This step will require the Python library "natsort" if it has not already been installed. Next, place and route also required a new strategy as timing should be considered during optimized placement. First the multi mode multi corners (mmmc) were created for optimized placement and the qrc file was downloaded from Github. Normally the foundry would be contacted for the mmmc details, however it is unnecessary at this moment, and only two corners were created. After this step has been completed, the design is optimized and then placed. Before routing, it is recommended to verify the connectivity and periodically conduct DRC checks to spot any underlying issues. Because the file was only routed globally, there were plenty of errors and connectivity issues present. These should be rectified after detailed routing and can be ignored at this stage. The subsequent procedure is to synthesize the clock tree, which is a standard part of any physical design flow. For this reason, it was attempted with the regfile.v even though it only has one clock. There were two errors regarding connectivity, but they were corrected using the command "nanoRoute". And finally, the detail route is completed.

Comparatively, the file currently used in the design flow is a rudimentary, clean register that is much more straightforward than the completed core design. Thus, adapting the actual SwitchMCU core to the development process will likely introduce even more significant drawbacks and difficulties.

ACKNOWLEDGMENT

We would like to express our sincerest gratitude to Professor Yu Kee Ooi for this interesting and valuable semester thus far, as well as both Joe Cavallaro and Peter Varman for their resources used by the physical design team. From Joe Cavallaro, the physical design flow framework and files, and from Peter Varman, the regfile.v used in each stage of the methodology.

REFERENCES

[1] Synopsys, "What is RISC-V? – How does it work?," *Synopsys*, https://www.synopsys.com/glossary/what-is-risc-v.html (accessed Mar. 13, 2025).

[2] Rice-MECE-Capstone-Projects, "SWITCHMCU: Switch MCU Repository," *GitHub*, https://github.com/Rice-MECE-Capstone-Projects/SwitchMCU (accessed Mar. 13, 2025).

[3] K. Sharma, "What is Physical Design," *VLSI - Physical Design For Freshers*, https://www.physicaldesign4u.com/2019/12/what-is-physical-design.html (accessed Mar. 13, 2025).

[4] VLSI Talks, "Floorplanning in Physical Design," *VLSI TALKS*, https://vlsitalks.com/physical-design/floorplan/ (accessed Mar. 13, 2025).

[5] AnySilicon, "Ultimate Guide: Clock Tree Synthesis," *AnySilicon*, https://anysilicon.com/clock-tree-synthesis/ (accessed Mar. 13, 2025).

[6] Synopsys, "PrimeTime: Static Timing Analysis," *Synopsys*, https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html (accessed Mar. 13, 2025).

[7] R. T. Edwards, "create lef library" *GitHub*, https://github.com/RTimothyEdwards/open_pdks/blob/master/common/create_lef_library.py (accessed Mar. 13, 2025).