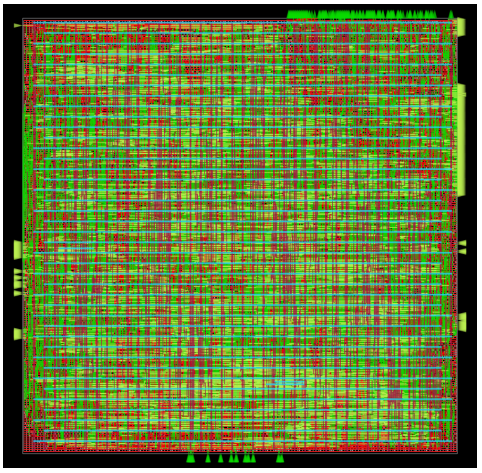
SHA-256 ASIC for Bitcoin Mining

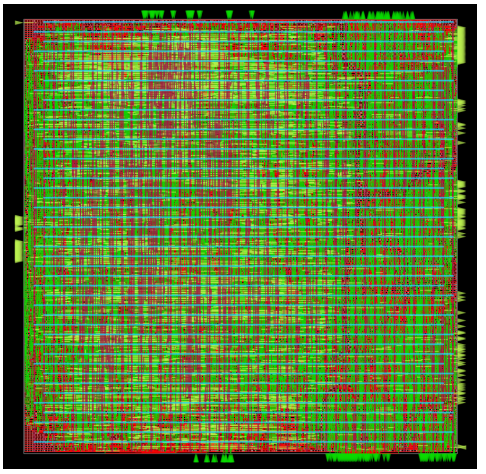
DEVASHISH GAWDE, ANISH MIRYALA, and NAVIN NADAR

1 CHIP LAYOUT

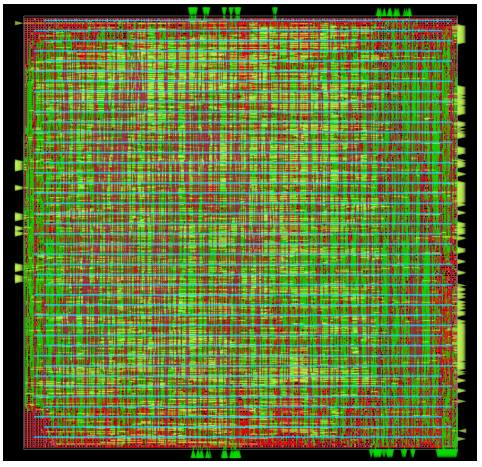
The provided figures represent the GDS-II layout files for the top module of the custom SHA256 hash function. This includes custom generation of the message padder, message scheduler, and the Naive core (compression function) modules. Each layout is the outcome of implementation with four types of adders: the '+' operator, Brent-Kung, Han-Carlson, and Kogge-Stone.



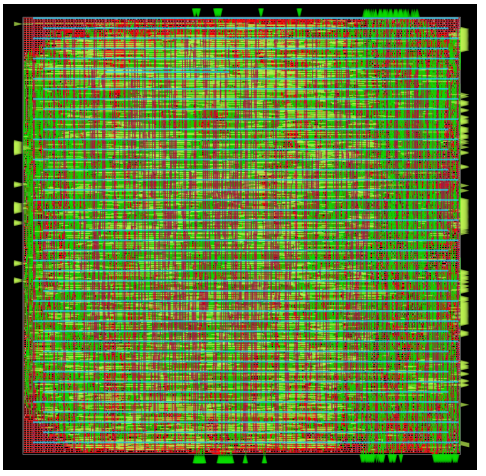
(a) '+' operator



(b) KS adder



(c) BK adder



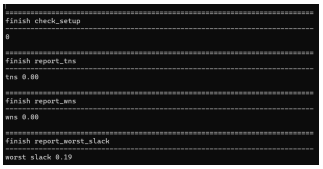
(d) HC adder

Fig. 1. GDS-II Layouts

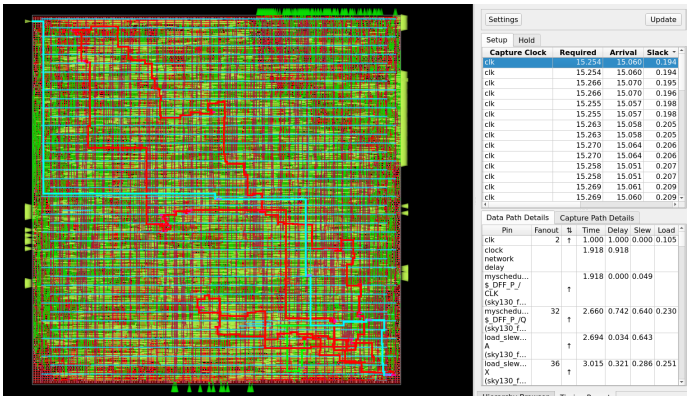
Authors' address: Devashish Gawde, dg4015@nyu.edu; Anish Miryala, am12553@nyu.edu; Navin Nadar, nn2446@nyu.edu.

2 STA TIMING REPORT

The Static Timing Analysis (STA) was conducted for all SHA-256 adder designs, initially uncovering timing violations. Through iterative adjustments to the design constraints, all violations were successfully resolved, resulting in zero setup and hold timing issues across all designs. The accompanying figures display snapshots from both the timing report obtained from OpenROAD and the worst path identified in the OpenROAD GUI.

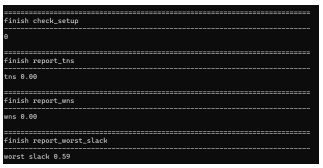


(a) Timing report

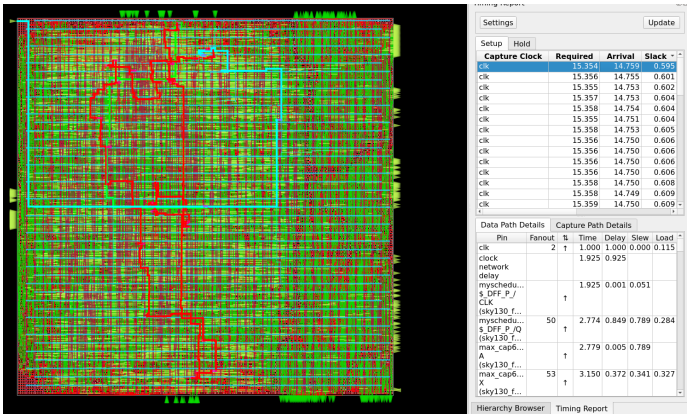


(b) Worst path

Fig. 2. STA '+' operator

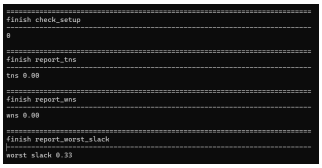


(a) Timing report

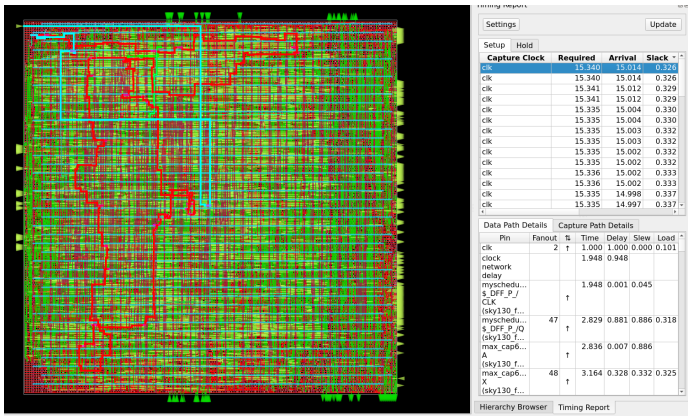


(b) Worst path

Fig. 3. STA KS adder

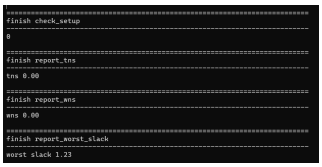


(a) Timing report

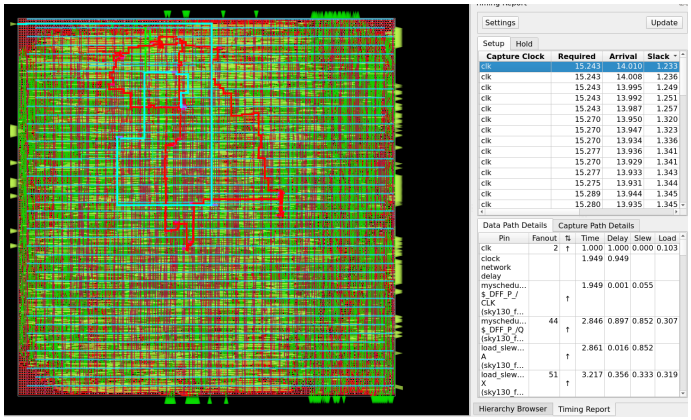


(b) Worst path

Fig. 4. STA BK adder



(a) Timing report



(b) Worst path

Fig. 5. STA HC adder

3 DRC REPORT

The DRC report was generated for all the adder designs of SHA-256. Initially, numerous DRC violations were observed, which were subsequently addressed and mitigated iteratively by the automated process of OpenRoad. Through these iterations, no DRC violations were detected for any of the designs created. Notably, the SHA256 Design employing the Kogge-stone adder exhibited the highest number of DRC violations, for which the DRC log file is showcased in the figure above. We see that "5_route_drc.rpt" is the final DRC report showcasing zero violations. This highlights how carefully checking for errors at each step ensures that our layout designs meets the manufacturing guidelines.

```
deva_shish11@DESKTOP-CH510G4:~/OpenROAD-flow-scripts/flow/reports/sky130hd/sha256top_ks/base$ ls
2_floorplan_final.rpt  4_cts_pre-repair.rpt                  5_route_drc.rpt         5_route_drc.rpt-5.rpt  cts_clk.webp          synth_check.txt
3_detailed_place.rpt   5_global_place.rpt                    5_route_drc.rpt-10.rpt  6_finish.rpt           final_clocks.webp     synth_stat.txt
3_resizer.rpt          5_global_route.rpt                    5_route_drc.rpt-15.rpt  VDD.rpt                final_ir_drop.webp
3_resizer_pre.rpt      5_global_route_post_repair_design.rpt 5_route_drc.rpt-20.rpt  VSS.rpt                final_placement.webp
4_cts_final.rpt        5_global_route_post_repair_timing.rpt 5_route_drc.rpt-25.rpt  antenna.log            final_resizer.webp
4_cts_post-repair.rpt  5_global_route_pre_repair_design.rpt  5_route_drc.rpt-30.rpt  congestion.rpt         final_routing.webp
deva_shish11@DESKTOP-CH510G4:~/OpenROAD-flow-scripts/flow/reports/sky130hd/sha256top_ks/base$
```

Fig. 6. Results showing multiple DRC reports

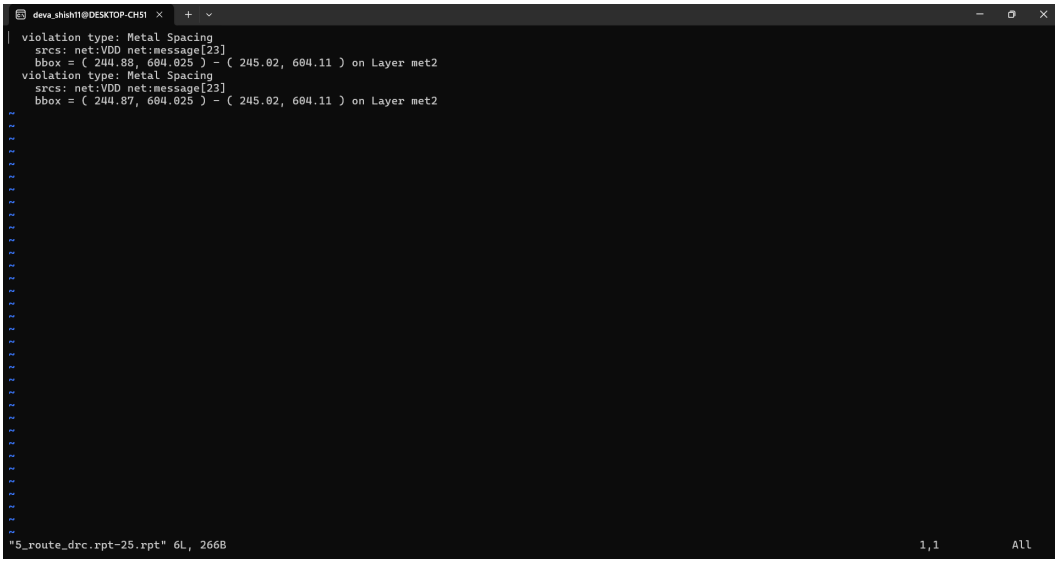


Fig. 7. Result showing DRC violations

Fig. 8. Final DRC Report with zero violations

4 SIMULATION WAVEFORMS

The figure shows the simulation waveform for the SHA256 top module running on several test cases. The hash is seen at output only when the hashing is complete and a done flag is raised, else remains 0. The test cases and expected results are read from a text file and the result is compared accordingly. The comparison can be checked at the display output (Here we used Xilinx Vivado).

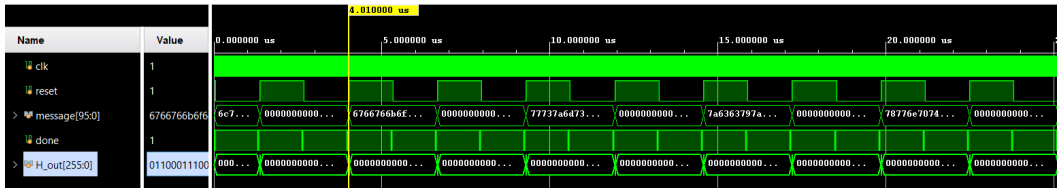


Fig. 9. Simulation waveform

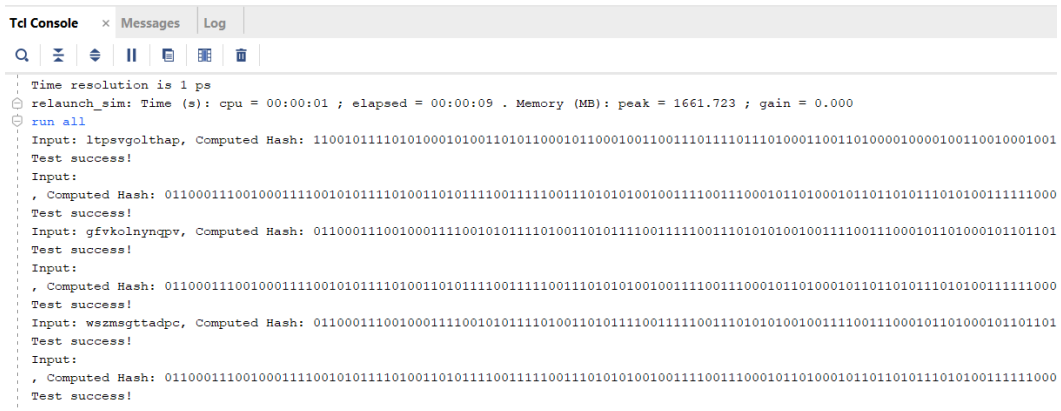


Fig. 10. Console output

5 ADDITIONAL WORK

5.1 Summary

As mentioned in the feedback, our team indeed gained significant knowledge about the Bitcoin algorithm and its architectures, but our experience extended far beyond that. The process of generating RTL code, selecting adders, and going through the physical design flow provided us with a comprehensive understanding of the entire design process. We acquired valuable technical skills in RTL design, adder analysis, physical design flow, and the use of industry-standard EDA tools.

Initially, we discovered that the GitHub repository contained incorrect code for the SHA-256 algorithm. Rather than simply reusing this flawed code, we took the initiative to clean up the files and attempted to make the algorithm work for all three architectures. When this approach proved ineffective, we decided to code the Naive and Pipeline architectures from scratch. To ensure the proper functioning of our newly coded architectures, we created comprehensive test benches. These test benches were designed to validate the correctness and performance of our implementations

correctly. This process required a significant investment of time and effort, as we had to develop and test each architecture independently.

In addition to coding the architectures from scratch, we introduced more modules to extend the functionality of the SHA-256 compression function: the message padder, message scheduler, and the SHA-256 core. The message padder ensures that the input message is properly formatted for hashing, while the message scheduler organizes the message into the required format for processing. The SHA-256 core is designed to compute multiple 512 blocks of data efficiently, depending on the input size. These modules extend the capabilities of the standard SHA-256 algorithm. The block diagram provided in the presentation illustrates the integration of these additional modules and highlights the overall architecture of our design.

Using industry-standard EDA tools like Xilinx Vivado, Cadence Virtuoso, Synopsys IC Compiler, KLayout, and OpenROAD significantly enhanced our digital design skills and knowledge. We gained practical experience in RTL design, custom layout, place and route, and design verification. This hands-on experience with both commercial and open-source tools broadened our understanding of the EDA landscape.

5.2 Investigating Yosys Adder Mapping Superiority

Our examination of the netlists generated by Yosys for both programs identified a higher number of adders than originally anticipated. This discrepancy suggests that Yosys might be incorporating additional adders internally to handle operations beyond the custom adders used in our design architecture.

If Yosys is indeed using these internal adders, it could explain the observed implementation results. The presence of these additional adders might influence the performance or resource utilization compared to the scenario where only the custom adders were used.

Yosys might have opted for a smaller and faster design based on the target library. Yosys analyzes the entire Verilog code and can potentially exploit specific characteristics of the design. Moreover, the tool Yosys uses for logic optimization (like ABC) might have additional optimizations for specific adder types in the target library. These optimizations could lead to a smaller or faster design compared to the custom-adder implementation.

5.3 LVS Challenges: Netlist Incompatibility Hurdles

In our project, we conducted Layout Versus Schematic (LVS) checks on our SHA256 design using KLayout. Initially, we executed the sample LVS scripts provided within the tool to familiarize ourselves with the process. Through this exercise, we discovered that these scripts could be customized to enhance the LVS testing procedure, based on guidance and examples from the KLayout website. However, we encountered significant challenges in generating the .cir file from Cadence Virtuoso, which is essential for running LVS checks on our designs. Exporting a .cir file, which is a SPICE netlist format required by KLayout for LVS checks, was not straightforward. This incompatibility required additional steps and potential use of intermediate tools or scripts to convert the netlist into the .cir format. Due to this difficulty, we were unable to perform the LVS checks on the designs we produced.