ECE 260B (WI25) Project Report

Dual Core Machine Learning Accelerator for Attention Mechanism

Team Details

Team Name: GroupNameGoesHere (team member names/emails/PIDs available in Appendix A)

Code Directory

All of our code can be found here:

`/home/linux/ieng6/ee260bwi25/shchopra/ECE260B_Project`

All directories referenced in the rest of this document will be subdirectories of this base directory in LinuxCloud.
Our project can also be found at the following GitHub repository:

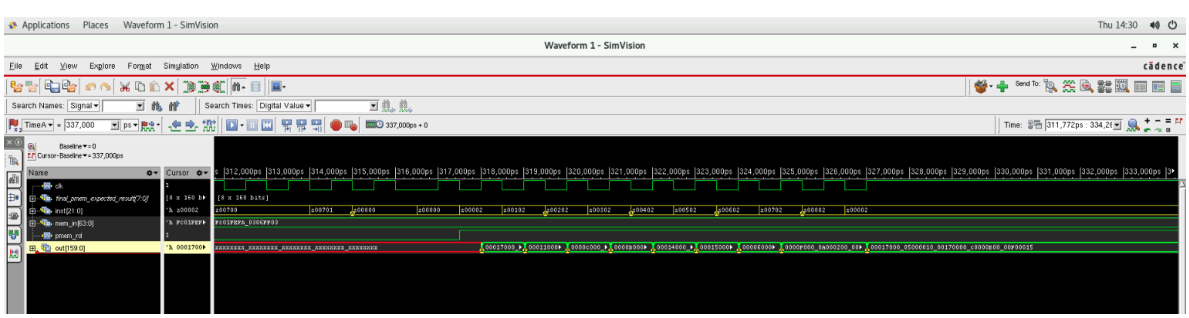
https://github.com/shchopraUCSD/ECE260B_Project

Please note:

- We first worked in the `single_core` directory. Once RTL, synthesis, PNR etc. were done for `single_core`, we then created a copy and shifted to `dual_core` directory
- Most of the final code for consideration for this project are in `dual_core` directory, unless mentioned otherwise (e.g. single core's `fullchip.v` and `fullchip_tb.v`, had to be re-defined for dual core).

Step 1: Single Core RTL, Synthesis, PNR

- RTL can be found in the following subdirectory: `dual_core/rtl/verilog/`
- New instructions were implemented for SFP division and accumulation.
- Synthesis was performed in this directory: `dual_core/synth/synth_without_sram` (to perform synthesis, the command is `dc_shell → source run_dc.tcl`). Note that this excludes SRAMs since we wish to do hierarchical PNR
- Aside: intermediate GLS
  - We also performed a synthesis run with SRAMs in the `single_core/synth/synth_with_sram` directory (same run command as before) - this was done to do GLS (gate-level sim) on a synthesized netlist as an intermediate step (while PNR efforts were ongoing) to fix X-prop bugs in our design. This GLS was done in the `single_core/gls/synth_with_sram` directory (command: `source run_gui`).
- PNR completed .enc file is here: `dual_core/pnr/core/route.enc`
- Final GLS on the PNR netlist was performed in this directory: `dual_core/gls/pnr_with_sram_pnr/` (see the files `run_gui`, `fullchip_tb.vcd`, `xrun.log` file for TB result with values matching). Launch command: `source run_gui` in this directory
- With our optimizations (more details in Step 5 section) we were able to meet WNS>=0 along with functional correctness in GLS at 1GHz frequency:



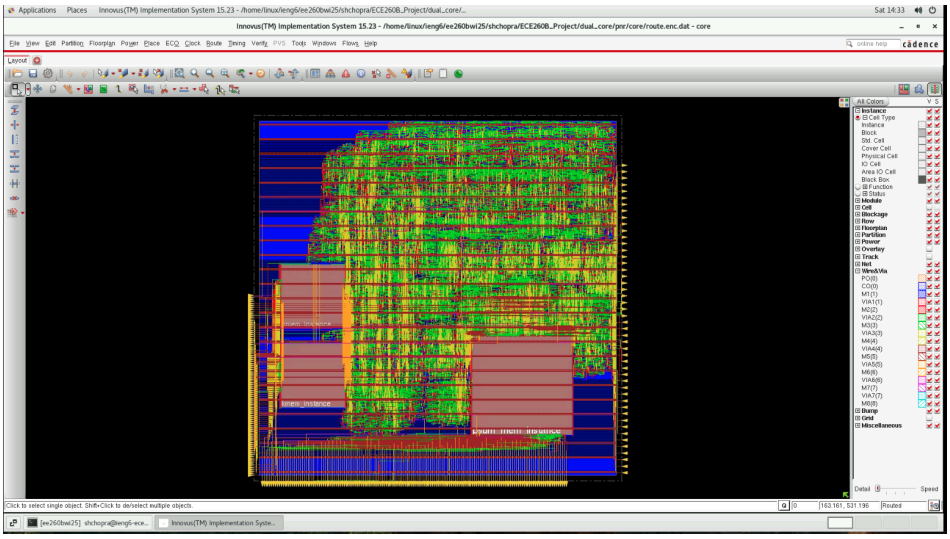
GLS Waveform with PNR netlist

Step 2: Output Normalization

- This is performed in the following file: `dual_core/rtl/verilog/sfp_row.v`
- Note that the RTL, PNR, and GLS sims reported in Step 1 already include the output normalization.
- our implementation sends the output of `mac_array` to the `ofifo`, the `ofifo` then sends a vector (when ready) to the `SFP`. The `SFP` then performs the normalization, and stores the normalized vector in `pmem`. In the TB, we fetch the vectors from `pmem`, and compare them with expected value vectors (computed in the TB itself).
- The SFP also has a pass-through mode (with a port in `sfp_row.v` called `pass_through`) that does not perform normalization, and instead passes the vector as it is to PMEM. This is used in the norm*V calculations.
- Note that the testbench `single_core/rtl/verilog/fullchip_tb.v` runs the design in both Q*K and norm*V calculation modes, performs estimated result computations, and then compares the final outputs (by reading them from PMEM) and comparing them with the estimated results. Please refer to the GLS commands with PNR netlist provided in Step-1.

Step 3: Hierarchical Synthesis & PNR of Single Core

- We implemented two different SRAM modules: `dual_core/rtl/verilog/sram_w8_64b.v` (for qmem and kmem) and `dual_core/rtl/verilog/sram_w8_160b.v` (for pmem)
- We then performed synthesis of both of them in the directories `dual_core/synth/synth_only_sram_w8_64b` and `dual_core/synth/synth_only_sram_w8_160b`
- Then, synthesis of the top module `core` for the single core, excluding the SRAMs' RTL, was performed, in `dual_core/synth/synth_without_sram`; this netlist was copied to PNR directory
- Then we moved to the PNR steps:
    - PNR of the SRAM submodules were done in `dual_core/pnr/pnr_sram_w8_64b` and `dual_core/pnr/pnr_sram_w8_160b`
    - PNR of the core, done hierarchically, with the SRAMs as submodules (lib files, gds, etc. copied to `./subckt` subdirectory)
- The RTL, path to PNR-completed .enc file, and waveform vcd of GLS are the same as Step 1. We achieved WNS>=0 and no DRC errors for SRAMs as well as for the entire single core.
- Note that we actually have two separate PNR runs for single core, one with the core as a top-module, performed in `single_core/pnr/core` directory, and another where the core is itself also a submodule, i.e. it exposes its power pins, and also has setMaxRouteLayer set to 7. This is done in the `dual_core/pnr/core` directory, and is the one that is finally used for Step-4.
- Running `source run_innovus.tcl` inside the innovus command prompt will perform the entire PNR, publish reports, perform DRC checks etc.

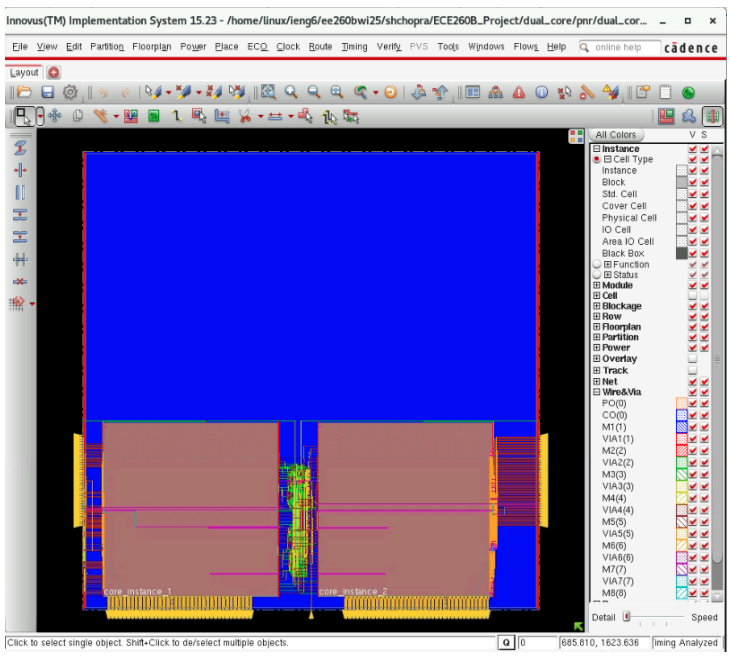


Single core PNR netlist result

- The waveform VCD and GLS snapshot of functional correctness have already been described in Step-1 of this document.

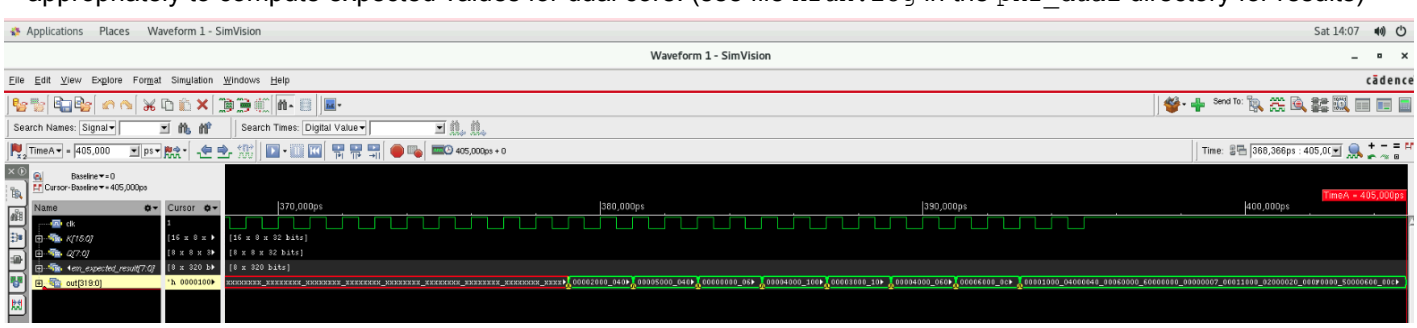
Step 4: Hierarchical Synthesis & PNR of Dual Core

- As mentioned previously, we performed PNR for the single core as a submodule (which itself used the SRAMs as a submodule) and used this single core as a submodule in dual core.
- To simplify controls over the two different clock domains (one per each core) we actually create two clock ports `clk1` and `clk2` in `fullchip.v`; both of these are driven by the same `clk` in the `fullchip_tb.v` testbench
- We implemented two bespoke async FIFOs using gray codes between the two cores to share the sum values, with `clk1` and `clk2` as the `rd_clk` and `wr_clk` for the first FIFO, and the other way around for the second FIFO. These shared sum values are used for normalization, and the TB controls the operations for both SFPs in a cycle-accurate manner.
- We chose async FIFOs over req/ack since the individual cores share the sum values frequently. Req/ack requires the values to be stable during the handshake, which would require the core to halt. Async FIFOs remove this requirement, and are also easier to use as a plug-and-play component given the design constraints.
- A new instruction was created and passed from the TB to control passing the sums between the two SFPs.
- The full RTL code can be found in the `dual_core/rtl/verilog/` directory. In particular, the `fullchip.v` file is where the two cores and the async FIFO are instanced.
- Synthesis for the dual-core `fullchip` as the top module (i.e. core not included) was performed in the `dual_core/synth/synth_fullchip_dual_core` subdirectory
- The PNR for dual core was performed in the `dual_core/pnr/dual_core` directory. The final PNR completed file is `route.enc`, and the PNR can be run by running `source run_innovus.tcl` in the innovus prompt. We were able to meet WNS>=0, and 0 DRC/connectivity/geometry violations.



Final Dual Core PNR Result

- GLS for the final dual core was performed with both the synthesis netlist and the PNR netlist, in the directories `dual_core/gls/synth_dual` and `dual_core/gls/pnr_dual`, respectively (`source run_gui`). TB was also modified appropriately to compute expected values for dual core. (see file `xrun.log` in the `pnr_dual` directory for results)



GLS with PNR netlist waveform

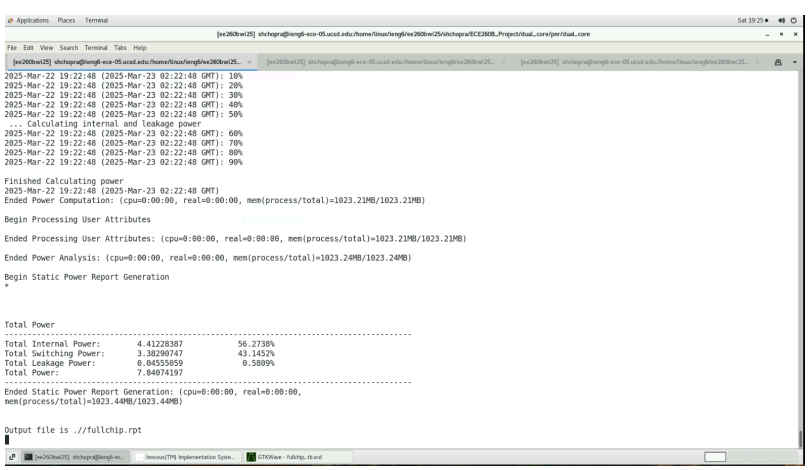
Step 5: Optimizations

We performed the following optimizations to our design (all results shown in previous steps already include these):

- RTL
  - We implemented a restoring division algorithm with pipelining in `sfp_custom_div.v` which is used in the `sfp_row.v` module
  - Note: for single core, we initially tried multi-cycle path for the SFP division, which was able to meet timing in synthesis but not in PNR. Hence, we switched to the custom division.
  - Pipelining
    - Custom pipelined multiplication, and pipelining of adder tree inside `mac_8in` module
    - Adder trees, and the division in `sfp_row.v` were also pipelined
    - Note that without pipelining, we were not able to meet timing even with a relaxed CLK of period 4ns.
  - To simplify timing, we moved the SFP between the OFIFO and the PMEM. That is, once the OFIFO has a vector ready, it passes the vector to the SFP, which performs the normalization, and then writes the normalized vector directly to PMEM.
- Placement
  - We also iterated on the dual core PNR to move around pins so that timing would be met.
  - For dual core, the single core instance on the right hand side is flipped about the Y-axis. This is done so that the core on the left side is driven by pins on the left side of the chip, and the core on the right side is driven by pins on the right side of the chip.
  - Output pins are always at the bottom edge, for both single core and dual core.
  - Pitch for output pins of single core and output pins of dual core have been matched.
- Miscellaneous
  - We were initially not meeting timing for the dual core, in the clk→Q path for the reg2reg path of single core module output to final dual chip module output. Earlier, there was a direct combinational path from the PMEM instances all the way to the dual core fullchip output pins. To solve this problem, we flopped the PMEM outputs at the single core boundary.
  - All synthesis was performed with worst corner libs. Setup timing for dual core was met with WC, and hold timing was met with BC. We also achieved 0 DRC/connectivity/geometry errors.

Conclusion

We were able to complete the design, meeting timing (both setup and hold) at the target frequency of 1GHz, with IO delays set as 0.2ns throughout. As per the reports, our total chip area is 3101825 um^2.
We also performed power analysis of our design in innovus/voltus, with the VCD from the GLS run that used the PNR netlist of the dual core fullchip. The total power came out to be 7.84 mW. Screenshots are below:



```
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 10%
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 20%
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 30%
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 40%
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 50%
 ... Calculating internal and leakage power
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 60%
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 70%
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 80%
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT): 90%

Finished Calculating power
2025-Mar-22 19:22:48 (2025-Mar-23 02:22:48 GMT)
Ended Power Computation: (cpu=0:00:00, real=0:00:00, mem(process/total)=1023.21MB/1023.21MB)

Begin Processing User Attributes

Ended Processing User Attributes: (cpu=0:00:00, real=0:00:00, mem(process/total)=1023.21MB/1023.21MB)

Ended Power Analysis: (cpu=0:00:00, real=0:00:00, mem(process/total)=1023.24MB/1023.24MB)

Begin Static Power Report Generation
*


Total Power
-----------------------------------------------------------------------------
Total Internal Power:        4.41228387            56.2738%
Total Switching Power:       3.38290747            43.1452%
Total Leakage Power:         0.04555059             0.5809%
Total Power:                 7.84074197
-----------------------------------------------------------------------------
Ended Static Power Report Generation: (cpu=0:00:00, real=0:00:00,
mem(process/total)=1023.44MB/1023.44MB)


Output file is .//fullchip.rpt
```

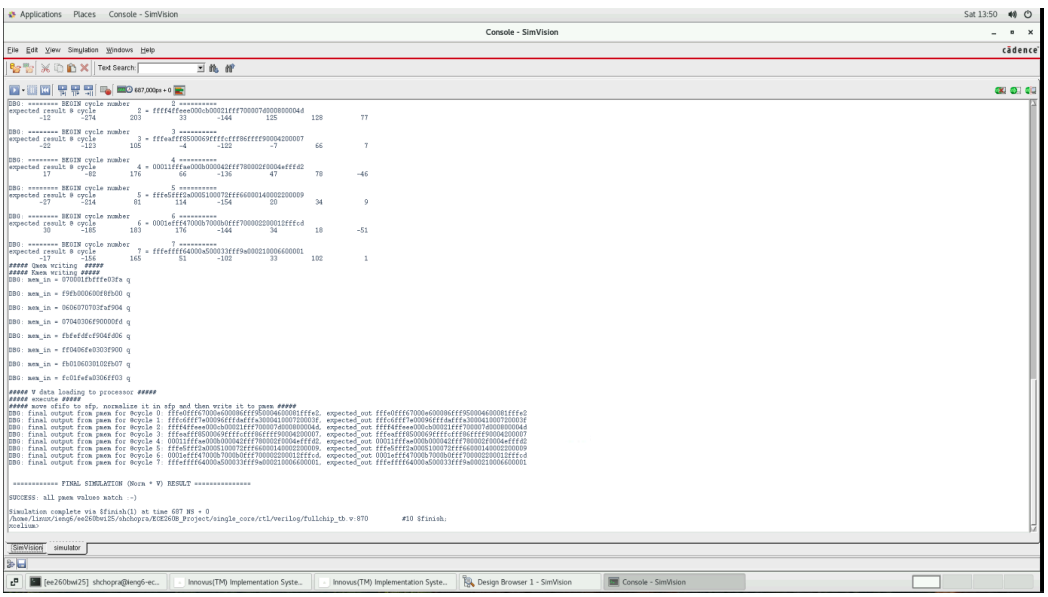
Innovus/Voltus Power Measurement with VCD - Results

Appendix A: Team Members

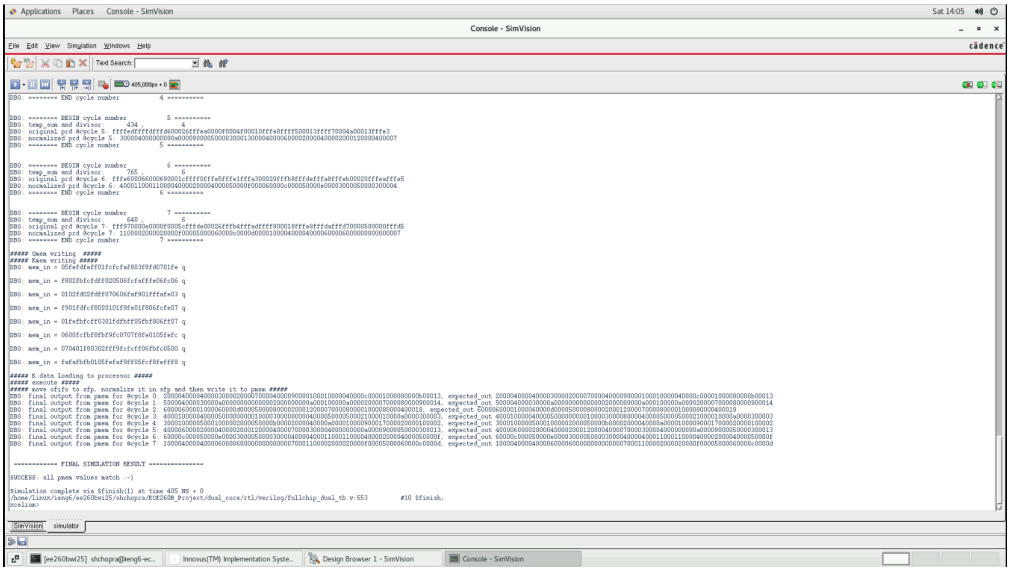
Please note: we had already checked with Prof. Kang about having 7 members in our team earlier in the quarter.

Name	**UCSD email (@ucsd.edu)**	**PID**
Shaurya Chopra	shchopra	A69029758
Mayank Kumar	mak025	A69030454
Divyang Wadhwani	dnwadhwani	A69030133
Navya Jain	n6jain	A69029808
Sidhartha Mishra	simishra	A69035835
Ishita Chawla	ichawla	A69029691
Soumil Paranjpay	sparanjpay	A69036087

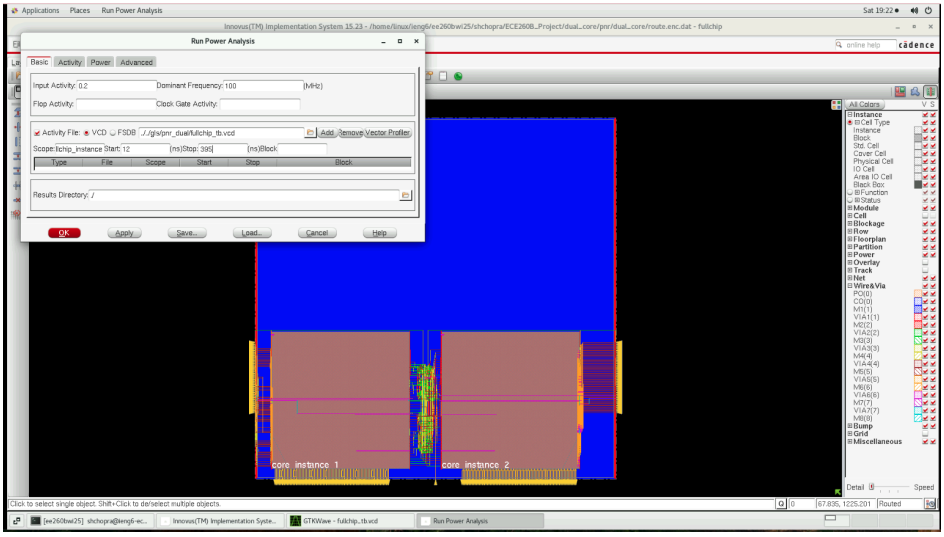
Appendix B: Useful Screenshots



GLS logs for Step-1 with PNR netlist that shows all values matching



GLS with PNR Netlist (Step-4) Log that shows all values matching



Innovus/Voltus Power Measurement with VCD for Dual Core - Setup