Week 10

Codefest: Hello doc(umentation)!

ECE 410/510
Spring 2025

Challenge #29

Overview and context:
Writing excellent documentation for a graduate-level chiplet design project requires balancing technical depth with clarity. The goal of this challenge is to provide some additional guidance and reminders on what a good documentation would look like. The key is treating documentation as a deliverable that demonstrates your technical competence and design thinking, not just an afterthought.

Learning goals:

- Write a good documentation for your chiplet design project.

Project Structure & Organization

- Your documentation should clearly demonstrate project success, as outlined at the beginning of the course (see Canvas:
  - You successfully benchmarked your SW algorithm and identified the bottleneck(s).
  - You designed, built, and tested (in software, e.g., Verilog) a custom HW accelerator chiplet to remove the bottlenecks.
  - You synthesized your HW design down to the ASIC/transistor level and obtained relevant performance metrics, e.g., max. frequency, number of transistors, etc.
  - You applied co-design and an iterative approach to improve your design throughout the term.
  - You evaluated and benchmarked your entire system, i.e., SW + HW + communication in-between and provided the evidence that your HW accelerator indeed accelerates the initial SW version.
- Top-level README should provide a clear project overview, installation instructions, and quick-start guide. Include your chiplet architecture diagram early - readers need to visualize the system before diving into details.
- Separate documentation by domain: Create distinct sections for the Python simulation/verification environment, Verilog RTL design, and the overall chiplet architecture. Each domain has different audiences and documentation needs.

Technical Content Requirements

- Architecture documentation should explain your chiplet partitioning rationale, inter-chiplet communication protocols, and how your design addresses specific AI/ML workload requirements. Include timing diagrams, protocol specifications, and design trade-offs you considered.
- Design decisions need thorough justification. Explain why you chose specific memory hierarchies, data paths, parallelization strategies, etc. Graduate-level work expects you to demonstrate an understanding of alternatives and trade-offs.
- Vibecoding promts must be included so that anybody can reproduce your results.

Code Documentation Standards

- Acknowledge the use of AI tools.

- Use tools like Sphinx for Python documentation and consider markdown with embedded diagrams for the overall project documentation. Keep your documentation version-controlled alongside your code, and write as you develop (as explained the beginning of this course) rather than retrofitting documentation at the end.
- Verilog modules should have comprehensive header comments explaining functionality, interface protocols, timing requirements, and any non-obvious implementation choices. Document your clock domains, reset strategies, etc.
- Python scripts need docstrings for all functions and classes, especially your testbenches and verification environments. Document your simulation methodology, what each test validates, and how to interpret results.

Hardware-Specific Elements

- Verification methodology documentation is crucial. Explain your testbench architecture, coverage metrics, and validation approach. Include waveform examples for key scenarios.
- Synthesis and implementation notes should cover your target technology, timing constraints, area/power results, and any optimization strategies you employed.

Academic Project Considerations

- Related work section should position your design relative to existing chiplet architectures and AI/ML accelerators. Graduate work expects engagement with current research. Use answers to the Heilmeier questions as a starting point.
- Results and analysis need quantitative evaluation. Include performance metrics (e.g., max. clock frequency, latency, throughput, etc.), resource utilization (e.g., number of transistors, power consumption, etc.), and comparisons to baseline implementations (i.e., your software implementation) or published results where possible.
- Future work should identify limitations and potential improvements, demonstrating deeper understanding of the problem space.