Christof Teuscher

ECE 410/510: Hardware for AI and ML

Codefest #1

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

teuscher Lab
teuscher-lab.com

Portland State
UNIVERSITY

Codefest

A codefest (a.k.a hackathon) is a collaborative event where participants, often in teams, work intensively on a project within a short timeframe to solve a problem or create a prototype.

Codefest goals

- **Rapid prototyping**, solve problems, and create complex designs.
- Learn how to use **LLMs** efficiently to create complex design we couldn’t create without.
- Use LLMs to learn and to gain a deeper understanding of the subject matter.
- Collaborate, share and learn from each other.
- Present, document, communicate.
- **Motivate you to go home and brings things to the next level.**

Today: Warm-up

- Explore “vibe coding”
- Install tools and get used to the workflow.
- Lay some groundwork for co-design.
- Start with high-level languages.
- Use profiling of different workloads to understand where CPU cycles are burnt and where the bottlenecks are.

Show your work

- Volunteer to show what you've got (whether it works or not)
- Google Meet link in Slack.

Week 2

Challenges

ECE 410/510
Spring 2025

Instructions:

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- Try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #6

1. Implement a simple neuron (a.k.a. perceptron) with two inputs and a sigmoid activation function. Hints: https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function
2. Use the perceptron learning rule (Google or LLM it) to train the neuron to realize the following binary logic functions:
    a. NAND
    b. XOR
3. Good video resources:
    a. A Gentle Introduction to Neural Networks: https://www.youtube.com/watch?v=b7oYqAlX_Bo
    b. But what is a neural network? https://www.youtube.com/watch?v=aircAruvnKk

Challenge #7

1. Visualize the learning process in a 2D-plane by representing the neuron's "line" that separates the space.
2. You can turn that in an animated visualization that illustrates every step of the weight updating process as you apply the perceptron rule.

Challenge #8

1. Implement a multi-layer feed-forward perceptron network. The network should have two input neurons, two hidden neurons, and one output neuron. Hints: https://machinelearningmastery.com/neural-networks-crash-course
2. Implement the backpropagation algorithm to train your network to solve the XOR logical function.

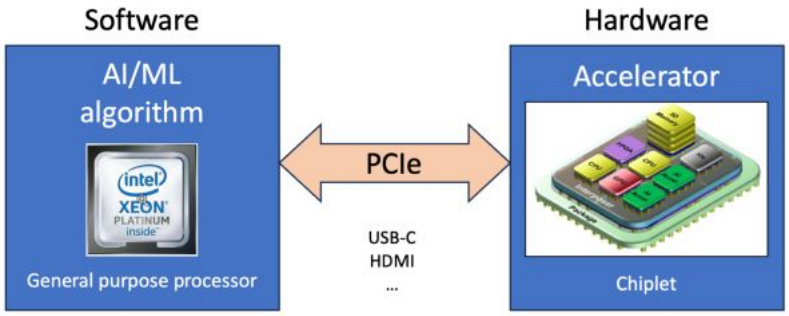
Week 2

Codefest: Bootstrapping the main project!

ECE 410/510
Spring 2025

Main project goals

- **Design, test, and benchmark a co-processor chiplet that accelerates parts of some AI/ML code/algorithm of your choice** (see list at the end). Start with a blank slate for your design.
- [ALTERNATIVE] Design a stand-alone chiplet. Your chiplet could, for example, include an ARM core.
- You will be deciding what design constraints to impose (e.g., power budget) and what metrics to use (e.g., throughput).
- Apply HW/SW co-design principles to decide what part of your algorithm is executed in HW.
- The chiplet should be described in a HW-description language (e.g., Verilog). The design should be synthesizable in order to obtain your relevant metrics.
- The deeper you can go into the HW design, the better. The ultimate goal is to go all the way down to an ASIC description. E.g., by using OpenLane's workflow to convert HDL into GDS: https://www.zerotoasiccourse.com/terminology/openlane.



Project dos and dont's

Dos

- Pick something exciting, at the forefront of technology
- Pick something that challenges you,
- Failure should be an option.
- Something new that you want to learn
- Start with a blank slate, but it's OK to draw inspiration from existing architectures.

Dont's

- No relying on current accelerators, e.g., GPUs
- No traditional CPUs (except for executing non-accelerated code).
- No cache algorithms, no branch predictors, etc.!
- No teamwork allowed. You must have your own project. If several want to work on the same AI/ML problem, you'd be competing with each other (in a friendly way!)

During codefests

- Practice "vibe coding" if you don't have the background to code on your own.
- Slack will be our main collaborative platform. Keep an eye on it.
- Post questions, solutions, insights in the `#codefests` channel.
- Present what you've got, whether it works or not.

Challenge #9

Learning goals:

- Understand the problem.
- Answer the Heilmeier questions.
- Analyze the algorithm, identify bottlenecks, generate data-flow graphs, profile the code, generate call graphs.
- Draw a high-level block diagram and/or flow chart of your algorithm (unless you can find one online).
- Get a first idea of what parts of your code would benefit from a HW acceleration.

Suggested tasks:

1. Answer the Heilmeier questions for your project idea:
   https://www.darpa.mil/about/heilmeier-catechism
2. If you have good answers to these questions, you likely have a solid project.
3. For question 2, use Google Scholar and ResearchRabbit.
4. Post your answers in the following Google Doc:
   https://docs.google.com/document/d/1fDuM5bUIuZBPGqjvlTHfYhtRlWyl6sWodc8ZgmDZwF4
5. Find code for your algorithm, write your own, or use "vibe coding." The code would ideally be in Python because of the availability of great tools and libraries. But any other high-level language that allows for profiling, data-flow analysis, etc., is fine too.
6. Establish an initial software benchmark that will serve as a baseline for comparing your accelerator HW later on.
7. Analyze the algorithm, identify bottlenecks, generate data-flow graphs, profile the code, generate call graphs. **For all of these tools, LLMs will be happy to provide sample codes**. You may want to use some of the following tools for that:
   - Python profiling tools like `cProfile`, `line_profiler`, or `py-spy` allow you to identify bottlenecks and optimize code for specific hardware.
   - `memory_profiler` can be used analyze memory usage patterns.
   - `Doxygen` can generate call graphs showing what functions/methods a particular function/method calls, which is a great way to understand the code flow. For your algorithm, this will visualize the relationships between different functions and methods. More at https://dzone.com/articles/understanding-code-call-graphs
   - `StaticFG` (Static Control Flow Graph) is a package that generates control flow graphs for Python 3 programs. While it focuses on control flow rather than data flow, these CFGs can be visualized with `Graphviz` and used as a foundation for static analysis, including data flow analysis.
   - `CodeQL` provides capabilities for data flow analysis in Python programs. It allows you to track the flow of data through a program to points where the data is used, with support for both local data flow (within a single method) and global data flow (throughout the entire program)
   - `PyFlowGraph` is a tool developed by IBM that can generate dataflow graphs from Python code, though it primarily performs dynamic analysis rather than static analysis. It outputs data in graph.ml format.
   - `PYCFG` is a tool that can be used to draw control flow graphs for Python code. While primarily for control flow, it can be extended to incorporate data flow information.
   - In Python, the Abstract Syntax Tree (`AST`) module is a powerful tool for analyzing and manipulating Python code programmatically. For creating custom dataflow analysis tools in Python, you can use:
     - Python's built-in ast module for parsing code into an abstract syntax tree
     - NetworkX for building and visualizing graph structures
     - Static analysis libraries like StaticFG for control flow graphs
8. Draw a high-level block diagram and/or flow chart of your algorithm.

   - You can either do that based on info about the algorithm you find online and/or
   - Based on the analysis and data of the tools above.
9. Get a first idea of what parts of your code would benefit from a HW acceleration.
   - `PyRTL`, `MyHDL`, or `pynq` can be used to design or interface with FPGAs or other hardware from Python.
   - `libusb`, `pyserial`, or `RPi.GPIO` can be used to interact with devices drivers through libraries.
   - `cocotb` or `MyHDL` can be used for Python-based hardware verification and co-simulation.

Typical AI/ML applications

- **Computer Vision**
  - Image classification and object detection
  - Facial recognition (with privacy considerations)
  - Medical image analysis
  - Quality control in manufacturing
  - Autonomous vehicles and robotics
- **Natural Language Processing**
  - Machine translation
  - Sentiment analysis
  - Text summarization
  - Chatbots and virtual assistants
  - Information extraction from documents
- **Recommendation Systems**
  - E-commerce product recommendations
  - Content recommendations (streaming services, news)
  - Advertisement targeting
  - Social media feed curation
- **Predictive Analytics**
  - Financial forecasting and risk assessment
  - Predictive maintenance for equipment
  - Supply chain optimization
  - Healthcare outcome prediction
  - Customer churn prediction
- **Anomaly Detection**
  - Fraud detection in finance
  - Network security monitoring
  - Manufacturing defect detection
  - Health monitoring systems
- **Time Series Analysis**
  - Stock market prediction
  - Energy load forecasting
  - Weather forecasting
  - Sales forecasting
- **Reinforcement Learning**
  - Game playing agents
  - Robotics control
  - Resource management optimization
  - Autonomous systems
- **Speech Recognition and Synthesis**
  - Voice assistants
  - Transcription services
  - Accessibility tools

  - Voice-based authentication
- **Generative AI**
  - Text generation (creative writing, code, content)
  - Image, video, and audio generation
  - Design assistance (architecture, fashion, graphics)
  - Synthetic data generation

Christof Teuscher

ECE 410/510: Hardware for AI and ML

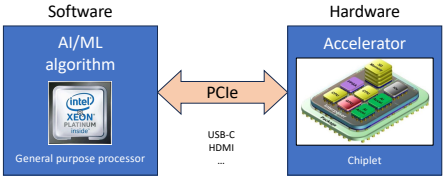
Codefest #2: Bootstrapping the final project

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

teuscher Lab teuscher-lab.com

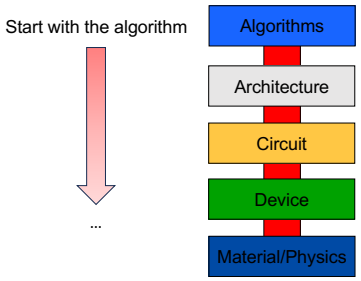
Portland State UNIVERSITY

Project vision



Design, test, and benchmark a co-processor chiplet that accelerates parts of some AI/ML code. Start with a blank slate for your design.
[Alternative]: design a stand-alone chiplet.

Compute stack



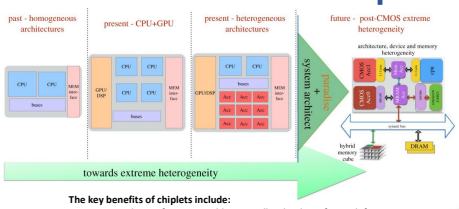
Example AI/ML applications

- **Computer Vision**
  - Image classification and object detection
  - Facial recognition (with privacy considerations)
  - Medical image analysis
  - Quality control in manufacturing
  - Autonomous vehicles and robotics
- **Natural Language Processing**
  - Machine translation
  - Sentiment analysis
  - Text summarization
  - Chatbots and virtual assistants
  - Information extraction from documents
- **Recommendation Systems**
  - E-commerce product recommendations
  - Content recommendations (streaming services, news)
  - Advertisement targeting
  - Social media feed curation
- **Predictive Analytics**
  - Financial forecasting and risk assessment
  - Predictive maintenance for equipment
  - Supply chain optimization
  - Healthcare outcome prediction
  - Customer churn prediction
- **Anomaly Detection**
  - Fraud detection in finance
  - Network security monitoring
  - Manufacturing defect detection
  - Health monitoring systems
- **Time Series Analysis**
  - Stock market prediction
  - Energy load forecasting
  - Weather forecasting
  - Sales forecasting
- **Reinforcement Learning**
  - Game playing agents
  - Robotics control
  - Resource management optimization
  - Autonomous systems
- **Speech Recognition and Synthesis**
  - Voice assistants
  - Transcription services
  - Accessibility tools
  - Voice-based authentication
- **Generative AI**
  - Text generation (creative writing, code, content)
  - Image, video, and audio generation
  - Design assistance (architecture, fashion, graphics)
  - Synthetic data generation

What is a chiplet?

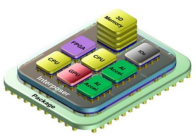
- A chiplet is a **small, specialized piece of silicon that can be combined with other chiplets to form a complete integrated circuit (IC) or system-on-chip (SoC).**
- Instead of manufacturing one large monolithic chip, the chiplet approach breaks down complex processor designs into smaller functional blocks that can be manufactured separately and then integrated together using advanced packaging technologies.
- Chiplets are connected using **high-speed, short-range communication interfaces** that enable them to work together as if they were on a single piece of silicon.
- This **modular approach** has been widely adopted by companies like AMD, Intel, and TSMC to create more efficient and powerful processors while overcoming the limitations of traditional monolithic chip design

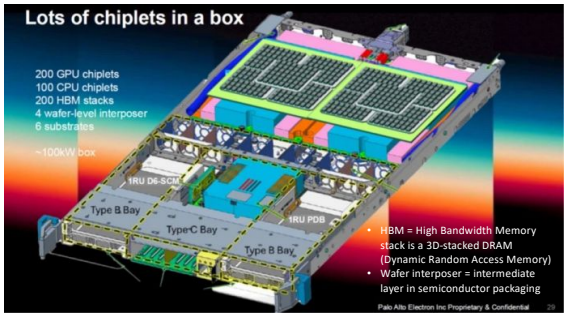
What is a chiplet?



The key benefits of chiplets include:

- Improved manufacturing yields, as smaller dies have fewer defects
- Better scalability for different performance and power requirements
- The ability to mix and match process technologies optimized for specific functions
- Reduced costs by reusing proven IP blocks across different products





https://semiengineering.com/chiplets-where-are-we-today

Communication interfaces

- **Advanced Interface Bus** (AIB): Developed by Intel, AIB is an open-source, die-to-die interconnect technology that enables high-bandwidth, low-power communication between chiplets. It supports both parallel and serial communication modes.
- **Universal Chiplet Interconnect Express** (UCIe): An industry-standard specification backed by major companies like AMD, Intel, Arm, and TSMC. UCIe aims to establish an open ecosystem for chiplet integration with standardized physical interfaces, protocols, and power management.
- **High Bandwidth Memory (HBM) Interface**: While primarily for connecting memory chiplets, this interface uses through-silicon vias (TSVs) and microbumps to achieve very high bandwidth connections.
- **Bunch of Wires (BoW):** A simpler, more cost-effective interface for medium-bandwidth connections between chiplets, developed through the Open Compute Project.
- TSMC's **Integrated Fan-Out (InFO)** and **Chip-on-Wafer-on-Substrate (CoWoS):** These are packaging technologies that include specific interconnect interfaces for connecting multiple chiplets.

Unclear of IP is available for any of these…

How to pick a project

Dos

- Pick something exciting, at the forefront of technology
- Pick something that challenges you
- Failure should be an option
- Something new that you want to learn
- Start with a blank slate, but it's OK to draw inspiration from existing architectures.

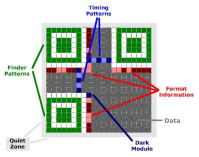
Dont's

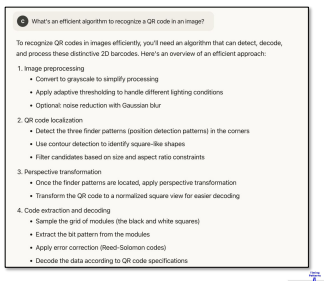
- No relying on current accelerators, e.g., GPUs
- No traditional CPUs.
- No cache algorithms, no branch predictors, etc.!
- No teamwork allowed. You must have your own project. If several want to work on the same problem, you'd be competing with each other (in a friendly way!).

Example

QR code recognizer for an edge device.

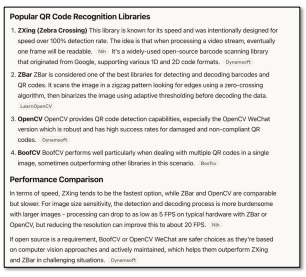
- Must be fast and ultra low-power.




What's an efficient algorithm to recognize a QR code in an image?

To recognize QR codes in images efficiently, you'll need an algorithm that can detect, decode, and process these distinctive 2D barcodes. Here's an overview of an efficient approach:

1. Image preprocessing
   - Convert to grayscale to simplify processing
   - Apply adaptive thresholding to handle different lighting conditions
   - Optional: noise reduction with Gaussian blur
2. QR code localization
   - Detect the three finder patterns (position detection patterns) in the corners
   - Use contour detection to identify square-like shapes
   - Filter candidates based on size and aspect ratio constraints
3. Perspective transformation
   - Once the finder patterns are located, apply perspective transformation
   - Transform the QR code to a normalized square view for easier decoding
4. Code extraction and decoding
   - Sample the grid of modules (the black and white squares)
   - Extract the bit pattern from the modules
   - Apply error correction (Reed-Solomon codes)
   - Decode the data according to QR code specifications


Popular QR Code Recognition Libraries

1. **ZXing (Zebra Crossing)** This library is known for its speed and was intentionally designed for speed over 100% detection rate. The idea is that when processing a video stream, eventually one frame will be readable. Nih It's a widely-used open-source barcode scanning library that originated from Google, supporting various 1D and 2D code formats. Dynamsoft
2. **ZBar** ZBar is considered one of the best libraries for detecting and decoding barcodes and QR codes. It scans the image in a zigzag pattern looking for edges using a zero-crossing algorithm, then binarizes the image using adaptive thresholding before decoding the data. LearnOpenCV
3. **OpenCV** OpenCV provides QR code detection capabilities, especially the OpenCV WeChat version which is robust and has high success rates for damaged and non-compliant QR codes. Dynamsoft
4. **BoofCV** BoofCV performs well particularly when dealing with multiple QR codes in a single image, sometimes outperforming other libraries in this scenario. BoofCv

Performance Comparison

In terms of speed, ZXing tends to be the fastest option, while ZBar and OpenCV are comparable but slower. For image size sensitivity, the detection and decoding process is more burdensome with larger images - processing can drop to as low as 5 FPS on typical hardware with ZBar or OpenCV, but reducing the resolution can improve this to about 20 FPS. Nih

If open source is a requirement, BoofCV or OpenCV WeChat are safer choices as they're based on computer vision approaches and actively maintained, which helps them outperform ZXing and ZBar in challenging situations. Dynamsoft

Heilmeier questions

1. **What are you trying to do**? Articulate your objectives using absolutely no jargon.
2. **How is it done today**, and what are the limits of current practice?
3. **What is new in your approach** and why do you think it will be successful?
4. Who cares? If you are successful, what difference will it make?
5. What are the risks?
6. How much will it cost?
7. How long will it take?
8. What are the mid-term and final "exams" to check for success?

https://www.darpa.mil/about/heilmeier-catechism

Week	Monday	Wednesday (Codefest)
2	HW/AI/ML overview + codesign overview	Start main project: pick workload, start analysis, benchmark, …
3	GPU architecture and programming for AI	Drafting a HW architecture, creating a model
4	Deep neural networks on GPUs	Coding HW description
5	Transformers on GPUs	First simulation + refinement
6	In-memory computation	Improving initial design
7	Neuromorphic chips: TrueNorth, Loihi, Akida	Simulation + refinement
8	Neuromorphic computing with mem-devices	Synthesizing design + benchmarking
9	Hardware accelerators for embedded systems	Final improvements
10	Emerging technologies and future directions	Final tests, validation, verification, benchmarking

Codefest goals

- **Rapid prototyping**, solve problems, and create complex designs using “vibe coding.”
- Learn how to use **LLMs** efficiently to create complex design we couldn’t create without.
- Use LLMs to learn and to gain a deeper understanding of the subject matter.
- Collaborate, share and learn from each other.
- Present, document, communicate.
- **Motivate you to go home and brings things to the next level.**

Week 3

Challenges

ECE 410/510
Spring 2025

Instructions:

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #10: Identify computational bottlenecks

1. Ask your favorite LLM to identity "computational bottlenecks" in the FrozenLake code from https://github.com/ronanmmurphy/Q-Learning-Algorithm
2. Do the suggestions make sense? How well is it able to identity bottlenecks?
3. Ask it to propose a HW implementation of the biggest bottleneck.
4. Ask it to generate System Verilog code for the HW implementation.

Challenge #11: GPU acceleration

5. Ask your favorite LLM to optimize the FrozenLake code from https://github.com/ronanmmurphy/Q-Learning-Algorithm for a GPU.
6. Benchmark both the pure Python and the GPU-accelerated versions and compare. How much speed-up do you get?

Week 3

Codefest: Decide the SW/HW boundary, pick tools, code toy example

ECE 410/510
Spring 2025

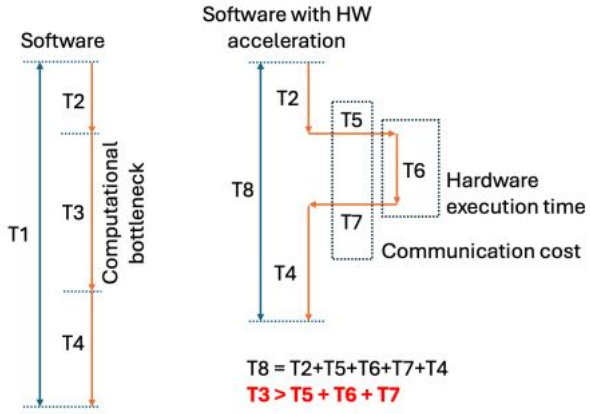
Challenge #12

Learning goals:

- Complete an estimate of the execution and communication times.
- Decide what parts stays in SW and what part becomes HW.
- Evaluate and pick a high-level tool for rapid prototyping and co-design.
- Implement a toy example to test the entire design workflow. E.g., FrozenLake RL code in which you implement the Q value updating formula in hardware.

Suggested tasks:

1. Before you complete the tasks below, you should benchmark and profile the algorithm that you chose. See codefest #2, challenge#9 for suggested tasks, tools, etc.
2. Based on your initial benchmarking and profiling of your code, make a first informed decision what part(s) should be accelerated in HW.
3. You may want to do a back-of-the-envelope calculation to see if the HW acceleration is worth the effort. To do so, determine the following times:



4. The HW acceleration is only worth it if T5 + T6 + T7 < T3.
5. Next, based on your background and interests, pick one of the high-level tools listed below that will allow you to do rapid HW prototyping in Python.
   - **PyMTL (Mamba):**
     - https://pymtl.github.io
     - User group: https://groups.google.com/g/pymtl-users
     - An open-source hardware modeling, generation, simulation, and verification framework.
     - MyHDL allows a subset of Python code to be translated to Verilog or VHDL. It offers co-simulation options where native Python code runs alongside a compiled simulation model of your hardware.
     - Hardware-software co-simulation using PyMTL3:
       - Create your hardware model in PyMTL3
       - Develop software that will interact with the hardware
       - Set up the co-simulation environment
       - Run and analyze results

- **PyRTL**:
  - https://ucsbarchlab.github.io/PyRTL
  - PyRTL is an open-source Python-based hardware development toolkit.
  - PyRTL provides a minimal set of hardware primitives, expressed as a Python class, which can then be extended with other classes and libraries as appropriate.
  - Allows for fast design iteration in a variety of domains, including cryptography and machine learning.
  - Paper:
    - Agile Hardware Development and Instrumentation with PyRTL
    - https://doi.org/10.1109/MM.2020.2997704

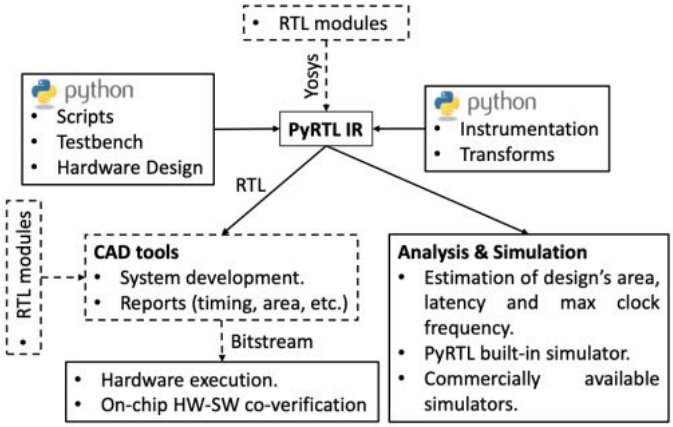


Fig. 1. Overview of PyRTL's flow.

- **Cocotb:**
  - https://www.cocotb.org
  - cocotb is an open source coroutine-based co-simulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.
- **MyHDL:**
  - https://www.myhdl.org
  - MyHDL turns Python into a hardware description and verification language, providing hardware engineers with the power of the Python ecosystem.
  - MyHDL designs can be converted to Verilog or VHDL.
- **pyUVM:**
  - https://github.com/pyuvm/pyuvm
  - pyuvm is the Universal Verification Methodology (UVM) implemented in Python instead of SystemVerilog. pyuvm uses cocotb to interact with simulators and schedule simulation events.
- **yosys:**
  - https://github.com/YosysHQ/yosys
  - Yosys is a framework for RTL synthesis and more. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application domains.

6. Implement a toy example to test the entire design workflow. E.g., FrozenLake RL code in which you implement the Q value updating formula in hardware (think adders + multipliers).
7. Once you know the tools will be right for you, start implementing your own HW design for your chosen algorithm.

Christof Teuscher
ECE 410/510: Hardware for AI and ML

Codefest #3: Decide the SW/HW boundary, pick tools, code toy example

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY

Next week

Monday
- CUDA
- Mapping deep neural nets (CNN) on GPUs
- TPUs

Wednesday
- No in-person codefest. Individual project time instead.
- Work from home or get together somewhere in small working groups.

Week 3 challenges

Challenge #10: Identify computational bottlenecks

1. Ask your favorite LLM to identity "computational bottlenecks" in the FrozenLake code from https://github.com/ronanmmurphy/Q-Learning-Algorithm
2. Do the suggestions make sense? How well is it able to identity bottlenecks?
3. Ask it to propose a HW implementation of the biggest bottleneck.
4. Ask it to generate System Verilog code for the HW implementation.

Challenge #11: GPU acceleration

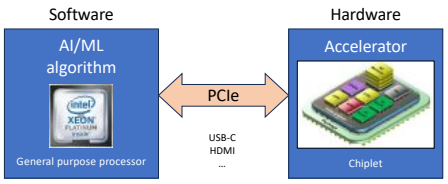
5. Ask your favorite LLM to optimize the FrozenLake code from https://github.com/ronanmmurphy/Q-Learning-Algorithm for a GPU.
6. Benchmark both the pure Python and the GPU-accelerated versions and compare. How much speed-up do you get?

Project page

Link on Canvas:

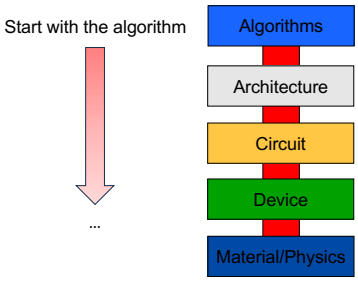
https://docs.google.com/document/d/1_HSDXhJEF1F2Qu76zJOlygvPm_F9NqX5VZ0f8CGXI34

Project vision



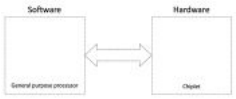
Design, test, and benchmark a co-processor chiplet that accelerates parts of some AI/ML code. Start with a blank slate for your design.
[Alternative]: design a stand-alone chiplet.

Compute stack



Main goals for today

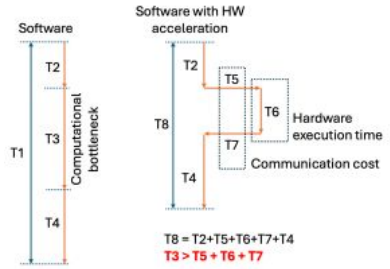
Pick a HW/SW boundary



Pick a workflow/toolchain

- PyMTL (Mamba)
- PyRTL
- Cocotb
- MyHDL
- pyUVM
- yosys
- …
- Candence
- Synopsis

Back-of-the-envelope estimate



Workflow/toolchain (1)

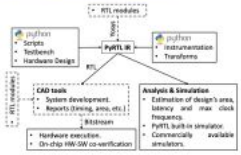
PyMTL (Mamba):

- https://pymtl.github.io
- User group: https://groups.google.com/g/pymtl-users
- An open-source hardware modeling, generation, simulation, and verification framework.
- MyHDL allows a subset of Python code to be translated to Verilog or VHDL. It offers co-simulation options where native Python code runs alongside a compiled simulation model of your hardware.
- Hardware-software co-simulation using PyMTL3:
  - Create your hardware model in PyMTL3
  - Develop software that will interact with the hardware
  - Set up the co-simulation environment
  - Run and analyze results

Workflow/toolchain (2)

PyRTL:

- https://ucsbarchlab.github.io/PyRTL
- PyRTL is an open-source Python-based hardware development toolkit.
- PyRTL provides a minimal set of hardware primitives, expressed as a Python class, which can then be extended with other classes and libraries as appropriate.
- Allows for fast design iteration in a variety of domains, including cryptography and machine learning.
- Paper:
  - Agile Hardware Development and Instrumentation with PyRTL
  - https://doi.org/10.1109/MM.2020.2997704



Workflow/toolchain (3)

Cocotb:

- https://www.cocotb.org
- cocotb is an open source coroutine-based co-simulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.

MyHDL:

- https://www.myhdl.org
- MyHDL turns Python into a hardware description and verification language, providing hardware engineers with the power of the Python ecosystem.
- MyHDL designs can be converted to Verilog or VHDL.

pyUVM:

- https://github.com/pyuvm/pyuvm
- pyuvm is the Universal Verification Methodology (UVM) implemented in Python instead of SystemVerilog. pyuvm uses cocotb to interact with simulators and schedule simulation events.

yosys:

- https://github.com/YosysHQ/yosys
- Yosys is a framework for RTL synthesis and more. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application domains.

Iterative, rapid prototyping codesign process

https://www.pacific-research.com/iterative-product-development

Communication interfaces

- **Advanced Interface Bus** (AIB): Developed by Intel, AIB is an open-source, die-to-die interconnect technology that enables high-bandwidth, low-power communication between chiplets. It supports both parallel and serial communication modes.
- **Universal Chiplet Interconnect Express** (UCIe): An industry-standard specification backed by major companies like AMD, Intel, Arm, and TSMC. UCIe aims to establish an open ecosystem for chiplet integration with standardized physical interfaces, protocols, and power management.
- **High Bandwidth Memory (HBM) Interface**: While primarily for connecting memory chiplets, this interface uses through-silicon vias (TSVs) and microbumps to achieve very high bandwidth connections.
- **Bunch of Wires (BoW):** A simpler, more cost-effective interface for medium-bandwidth connections between chiplets, developed through the Open Compute Project.
- TSMC's **Integrated Fan-Out (InFO)** and **Chip-on-Wafer-on-Substrate (CoWoS):** These are packaging technologies that include specific interconnect interfaces for connecting multiple chiplets.

Unclear of IP is available for any of these…

Week 4

Challenges

ECE 410/510
Spring 2025

Instructions:

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the #weekly-challenges Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #13: Benchmarking different SAXPY problem sizes.

Learning goals:

- Set up a CUDA development environment.
- Modify existing CUDA code.
- Profile and optimize your implementation.
- Compare and visualize the performance as you increase the problem size.

Tasks:

- Download the example code from https://developer.nvidia.com/blog/easy-introduction-cuda-c-and-c
- Modify the code so that you can simulate and measure the execution times for the following matrix sizes: $N = 2^{15}, 2^{16}, \ldots 2^{25}$ (or as high as you can).
- Visualize the execution times in a bar plot. You can use a spreadsheet or Matplotlib for that. What do you observe?
- If you want to make this even fancier, measure the total execution time (including memory transfers and allocations and the kernel execution time only (just the GPU computation) separately. Hint: `cudaEvent`
- If you don't have access to a GPU, you could use Google Colab:
  - https://colab.google
  - The free version of Colab gives you access to NVIDIA Tesla K80 GPUs.
  - https://www.nvidia.com/en-gb/data-center/tesla-k80

Challenge #14: Fibonacci sequence in CUDA

- Write a CUDA kernel that computes the Fibonacci sequence for *N* numbers. E.g., $N = 2^{20}$.
- The Fibonacci sequence is a famous mathematical sequence where each number is the sum of the two preceding ones. It starts with 0 and 1, and continues as follows: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
- The mathematical definition is:
  - $F(0) = 0$
  - $F(1) = 1$
  - $F(n) = F(n-1) + F(n-2)$ for $N-2 \geq n > 1$
- Compare a simple sequential implementation with your CUDA implementation. What do you observe?

Week 4

Codefest: Let's cook up some hardware

ECE 410/510
Spring 2025

Challenge #15

Overview and context:
The goals of last week's project challenge #13 were to (1) make a first attempt to decide your SW/HW boundary and to (2) test the entire design workflow by implementing a toy example. The goal for this week is to implement and simulate (in a high level HW description language) a version of your design.

Learning goals:

- Generate a first hardware description (in Verilog, SystemVerilog, or VHDL) of your accelerator chip (or of a toy example).
- Learn how to write HDL (or use vibe coding).
- Learn how to write a testbench.
- Learn how to simulate HDL.

Suggested tasks:

1. Based on your HW/SW boundary (which may be tentative) from last week, write an HDL (= hardware description language) description of your HW part.
2. If you have never done Verilog, SystemVerilog, or VHDL, here are some resources:
    - Verilog: https://www.geeksforgeeks.org/getting-started-with-verilog
    - SystemVerilog: https://verificationguide.com/systemverilog/systemverilog-tutorial
    - VHDL: https://nandland.com/learn-vhdl
3. Alternatively, take your Python or C/C++ high-level code and ask your favorite LLM to translate it into SystemVerilog.
4. Alternatively, use toy example (e.g., Frozen Lake Q table update formula) and convert that into a HW design (manually or with an LLM).
5. Write (or have your LLM write) a basic testbench for your design: https://fpgatutorial.com/how-to-write-a-basic-verilog-testbench
6. Simulate and test the design.
7. List of HDL simulators: https://en.wikipedia.org/wiki/List_of_HDL_simulators
8. Unless you know Cadence & Synopsis tools, I'd recommend you use verilator:
    - Verilator is a free Verilog/SystemVerilog simulator.
    - https://github.com/verilator/verilator
9. Alternatively, you can use one of the tools from the week 3 codefest (challenge #13) if you are able to write Verilog RTL code.

Week 5

Challenges

ECE 410/510
Spring 2025

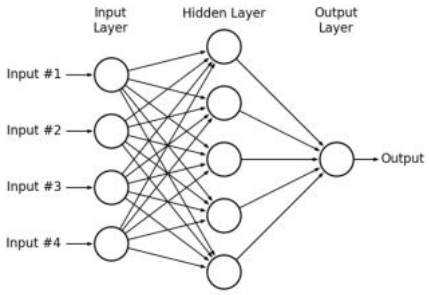
Instructions

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #16: Benchmarking SAXPY with PyTorch

Learning goals:

- Compare the performance of a simple feed-forward neural network (as seen in class) accelerated with CUDA vs accelerated with PyTorch.



Tasks:

1. (Vibe) Code a CUDA-accelerated version of a simple multi-layer feedforward, e.g., 4 inputs, 5 hidden neurons, 1 output, fully connected (as seen in class).
2. (Vibe) Code the same network by using PyTorch.
3. Benchmark both implementations and compare. What can you conclude?
4. If you want to go further, increase the depth and the width of the network and compare its execution time for various sizes. What's the outcome? Can you beat PyTorch with CUDA? Or vice versa?

Challenge #17: Sorting on a systolic array

Learning goals:

- Learn how to implement Bubble sort on a systolic array.
- Evaluate its performance as a function of the problem size.

Tasks:

1. Design a systolic array that can do Bubble sort. What dimension does the array need to have?
2. (Vibe) Code a software version in your favorite language and test it.
3. Visualize the execution times for various sorting sizes. E.g., 10, 100, 1000, 10000, etc.

Week 5

Codefest: Going down to physical/transistors (or CLBs) level

ECE 410/510
Spring 2025

Challenge #18

Overview and context:
The goal for this week is to take your HW accelerator design described in a high level HW description language (challenge #15) and to turn it into a physical design by using open-source EDA tools.

Learning goals:

- Synthesize the HW description you generated last week into a physical design description (transistor-level and/or FPGA)
- Install and learn how to use the OpenLane tools.

Suggested tasks:

1. Read through the OpenLane 2 "Newcomers" intro at https://openlane2.readthedocs.io/en/latest/getting_started/newcomers/index.html
2. Getting started with OpenLane 2
    - To check out an example of an OpenLane 2-based flow right in your browser, try the Google Colab™ notebook at https://colab.research.google.com/github/efabless/openlane2/blob/main/notebook.ipynb
    - To set up OpenLane 2 on your computer, check out the Getting Started guide at the following link: https://openlane2.readthedocs.io/en/latest/getting_started/index.html
3. Pick an example design, your own, or one generate by vibe-coding.
4. Try to obtain the maximum operating frequency for your ASIC design.
5. Alternative:
    - Use AMD's Vivado Design Suite to synthesize your design for an FPGA.
    - https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado/vivado-buy.html
    - Obtain the maximum operating frequency for your FPGA design.
6. Use the maximum operating frequency of your design to get a first estimate of the throughput of your accelerator chiplet.

Christof Teuscher
ECE 410/510: Hardware for AI and ML

Codefest #5: Going down to physical/transistors (or CLBs) level

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

Vibe Coding is not an excuse for low-quality work

A field guide to responsible AI-assisted development

"Move faster and break even more things."

I Am Devloper @iamdevloper

vibe coding, where 2 engineers can now create the tech debt of at least 50 engineers

https://addyo.substack.com/p/vibe-coding-is-not-an-excuse-for

Vibe coding

Educational study and paper

ECE 410/510, Spring 2025
Last update: Apr 28, 2025

Goal

Collaboratively write a journal article about new ways of teaching emerging technologies, especially computer engineering, AI, and ML. This will in part be a self-experiment. Specific questions we will attempt to answer:

- What are best practices?
- What worked? What didn't?
- How can the experience be replicated elsewhere?
- How to keep up with fast-moving new technology?
- Teaching foundations vs applied methods and tools?
- What does industry want?
- How to best use LLMs in teaching emerging technology
- Does self-grading work in this context?

Expected commitments and contributions

Authorship in a journal article must be earned with a significant contribution. If you want to become a co-author of this paper, you'd be expected to commit to and contribute the following things:

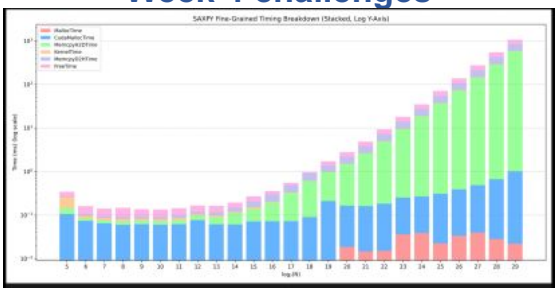
- Complete the two surveys (pre and post).
- Participate in the two focus groups.
- Assist with analysing Github code and portfolios (with a rubric)
- Write a section of the article.
- Assist with analyzing data and generating visualizations.

Timeline

- Mar 31 - Jun 6: Course
- May 6: Focus group #1
- Jun 9: Administer post-class survey
- Jun 17: Focus group #2
- Summer: write article
- Fall: submit article

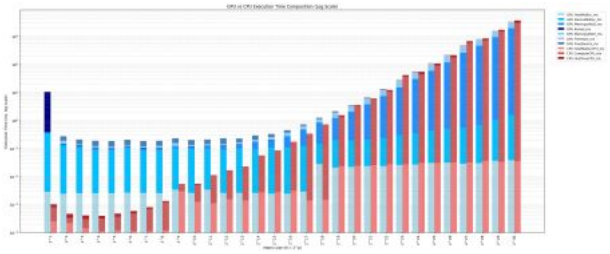
Announcement to be shared...stay tuned

Week 4 challenges



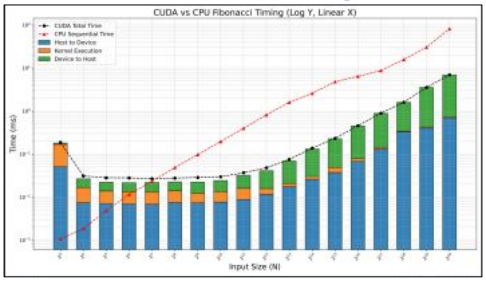
Source: Stephen Weeks

Week 4 challenges



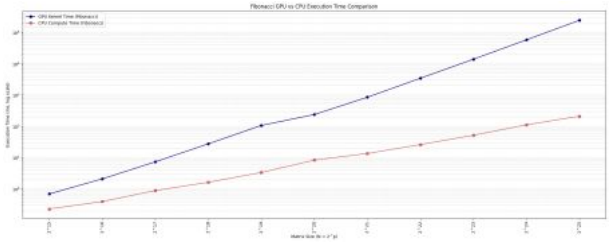
Source: Eric Zhou

Week 4 challenges



Source: Stephen Weeks

Week 4 challenges



Source: Eric Zhou

Fibonacci

Matrix Formulation of Fibonacci

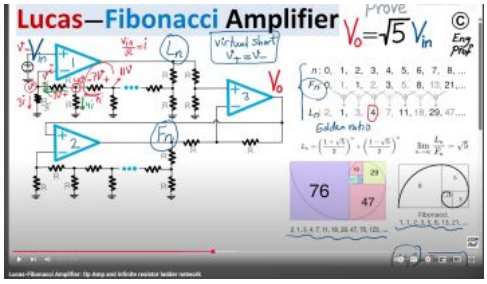
The Fibonacci sequence can be expressed using a 2×2 matrix:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

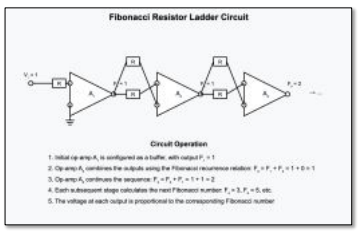
This means that raising the matrix to the power of n will give us the (n+1)th and nth Fibonacci numbers.

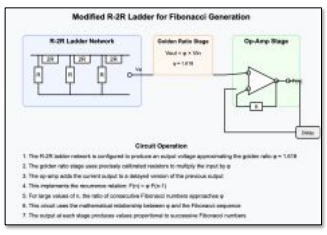
Parallel Matrix Exponentiation

1. **Divide and Conquer Approach**:
   - To calculate M^n, we can use the fact that M^n = M^(n/2) × M^(n/2) when n is even
   - And M^n = M^(n/2) × M^(n/2) × M when n is odd

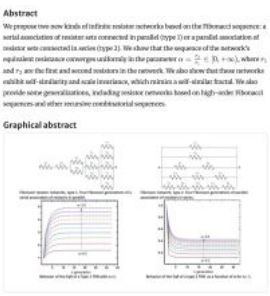


https://www.youtube.com/watch?v=jd_tWsTeLMY





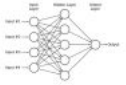
Thanks Claude...



Challenge #16: Benchmarking SAXPY with PyTorch

Learning goals:

- Compare the performance of a simple feed-forward neural network (as seen in class) accelerated with CUDA vs accelerated with PyTorch.



Tasks:

1. (Vibe) Code a CUDA-accelerated version of a simple multi-layer feedforward, e.g., 4 inputs, 5 hidden neurons, 1 output, fully connected (as seen in class).
2. (Vibe) Code the same network by using PyTorch.
3. Benchmark both implementations and compare. What can you conclude?
4. If you want to go further, increase the depth and the width of the network and compare its execution time for various sizes. What's the outcome? Can you beat PyTorch with CUDA? Or vice versa?

Challenge #17: Sorting on a systolic array

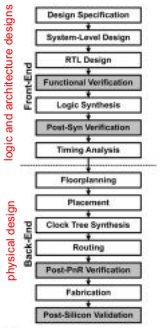
Learning goals:

- Learn how to implement Bubble sort on a systolic array.
- Evaluate its performance as a function of the problem size.

Tasks:

1. Design a systolic array that can do Bubble sort. What dimension does the array need to have?
2. (Vibe) Code a software version in your favorite language and test it.
3. Visualize the execution times for various sorting sizes. E.g., 10, 100, 1000, 10000, etc.

ASIC design flow



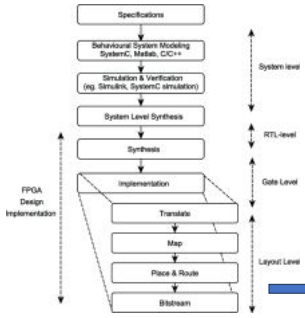
Python
Verilog/SystemVerilog/VHDL

Max. frequency, # transistors, etc.

Mishra, Ashutosh; Cha, Jaekwang; Park, Hyunbin; Kim, Shiho. Artificial Intelligence and Hardware Accelerators.

Fig. 9 ASIC design flow

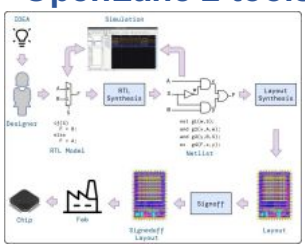
FPGA design flow

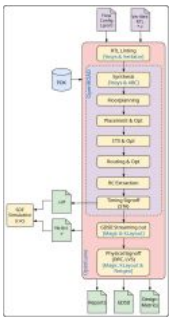


Max. frequency, # CLBs, etc

https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado/vivado-buy.html

OpenLane 2 tools





https://efabless.com/openlane
https://github.com/efabless/openlane2
https://openlane2.readthedocs.io/en/latest/getting_started/newcomers/index.html

Main goals for today

1. Check out the OpenLane 2 tools.
2. Go to the physical level with a sample design (vibe-generated) or your own initial HW design.
3. Ready yourself for design iterations & rapid prototyping.

https://www.pacific-research.com/iterative-product-development

Getting started with OpenLane 2

To check out an example of an OpenLane 2-based flow right in your **browser**, try the Google Colab™ notebook at
https://colab.research.google.com/github/efabless/openlane2/blob/main/notebook.ipynb

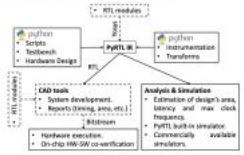
To set up OpenLane 2 on your **computer**, check out the Getting Started guide at the following link: https://openlane2.readthedocs.io/en/latest/getting_started/index.html

Workflow/toolchain (1)

PyMTL (Mamba):
- https://pymtl.github.io
- User group: https://groups.google.com/g/pymtl-users
- An open-source hardware modeling, generation, simulation, and verification framework.
- MyHDL allows a subset of Python code to be translated to Verilog or VHDL. It offers co-simulation options where native Python code runs alongside a compiled simulation model of your hardware.
- Hardware-software co-simulation using PyMTL3:
  - Create your hardware model in PyMTL3
  - Develop software that will interact with the hardware
  - Set up the co-simulation environment
  - Run and analyze results

Workflow/toolchain (2)

PyRTL:
- https://ucsbarchlab.github.io/PyRTL
- PyRTL is an open-source Python-based hardware development toolkit.
- PyRTL provides a minimal set of hardware primitives, expressed as a Python class, which can then be extended with other classes and libraries as appropriate.
- Allows for fast design iteration in a variety of domains, including cryptography and machine learning.
- Paper:
  - Agile Hardware Development and Instrumentation with PyRTL
  - https://doi.org/10.1109/MM.2020.2997704



Workflow/toolchain (3)

Cocotb:
- https://www.cocotb.org
- cocotb is an open source coroutine-based co-simulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.

MyHDL:
- https://www.myhdl.org
- MyHDL turns Python into a hardware description and verification language, providing hardware engineers with the power of the Python ecosystem.
- MyHDL designs can be converted to Verilog or VHDL.

pyUVM:
- https://github.com/pyuvm/pyuvm
- pyuvm is the Universal Verification Methodology (UVM) implemented in Python instead of SystemVerilog. pyuvm uses cocotb to interact with simulators and schedule simulation events.

yosys:
- https://github.com/YosysHQ/yosys
- Yosys is a framework for RTL synthesis and more. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application domains.

Week 6

Challenges

ECE 410/510
Spring 2025

Instructions

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #19: Implement a binary LIF neuron

Learning goals:

- Learn how a binary Leaky Integrate-and-Fire (LIF) neuron works.
- Implement such a neuron in Verilog or any other HDL.

Binary LIF neuron:
The binary LIF neuron can be formulated as:

- State representation: The neuron's state S(t) is either 0 (not spiking) or 1 (spiking)
- Simplfied update rule:
  - Accumulate input: $P(t) = \lambda P(t-1) + I(t)$
  - Where $P(t)$ is a potential variable, λ is a leak factor (between 0 and 1)
  - $I(t)$ is the binary input at time t
- Threshold function:
  - $S(t) = 1$ if $P(t) \geq \theta$ (threshold)
  - $S(t) = 0$ otherwise
- Reset mechanism:
  - If $S(t) = 1$, then $P(t)$ is reset to a lower value

Tasks:

1. Write a Verilog implementation of a simple binary LIF neuron (using the formulation above) with a single input.
2. Write a testbench that demonstrates the following scenarios:
   - Constant input below threshold
   - Input that accumulates until reaching threshold
   - Leakage with no input
   - Strong input causing immediate spiking

Challenge #20: Crossbar matrix-vector multiplication

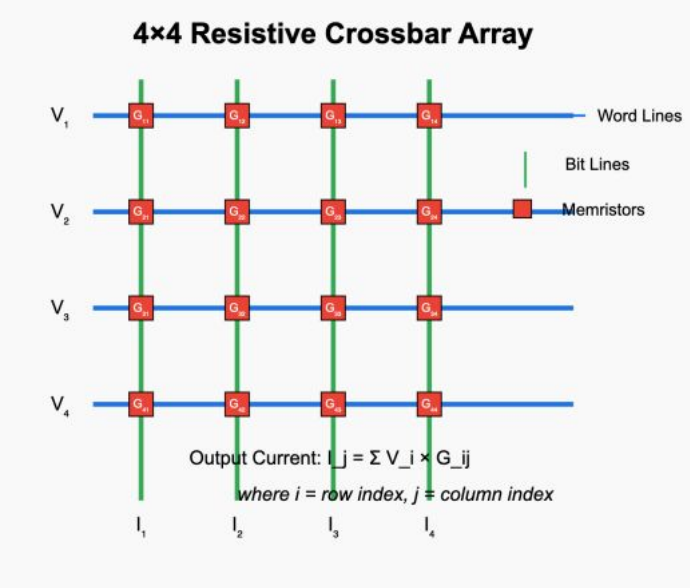
Learning goals:

- Learn how to simulate a resistive crossbar in SPICE
- Learn how matrix-vector multiplication in a resistive crossbar works

Tasks:

1. Write SPICE code for a 4x4 resistive crossbar (with fixed resistances)

2. Demonstrate that the resulting output currents represent the product of the 4x1 input vector and the 4x4 weight matrix.



Week 6

Codefest: Practice HW/SW co-design with design iterations

ECE 410/510
Spring 2025

Challenge #21

Overview and context:
The goal for this week is go back to the drawing board and to do a design iteration using your workflow.

Learning goals:

- Re-consider your initial HW/SW boundary
- Re-evaluate the bottlenecks
- Apply co-design principles

Things to (re-)consider and think about:

Workload analysis and characterization

- Profile/benchmark your algorithm again to understand computation patterns
- Identify key operations (matrix multiplications, convolutions, etc.)
- Analyze memory access patterns and data locality requirements
- Quantify performance bottlenecks in existing solutions

Architecture exploration

- Evaluate trade-offs between different accelerator architectures (systolic arrays, dataflow, etc.)
- Simulate performance with different memory hierarchies
- Consider scalability across different model sizes and types
- Explore different dataflow models (weight-stationary, output-stationary)

Microarchitecture design iteration

- Start with high-level models to validate architectural concepts
- Gradually refine RTL models with increasing fidelity
- Implement critical paths first to identify timing challenges
- If available, use emulation platforms (e.g., FPGA prototypes) for early validation

Progressive physical design

- Perform early performance/power analysis to guide microarchitecture decisions
- Iterate on critical paths identified through static timing analysis
- Use hierarchical design approaches for complex accelerators

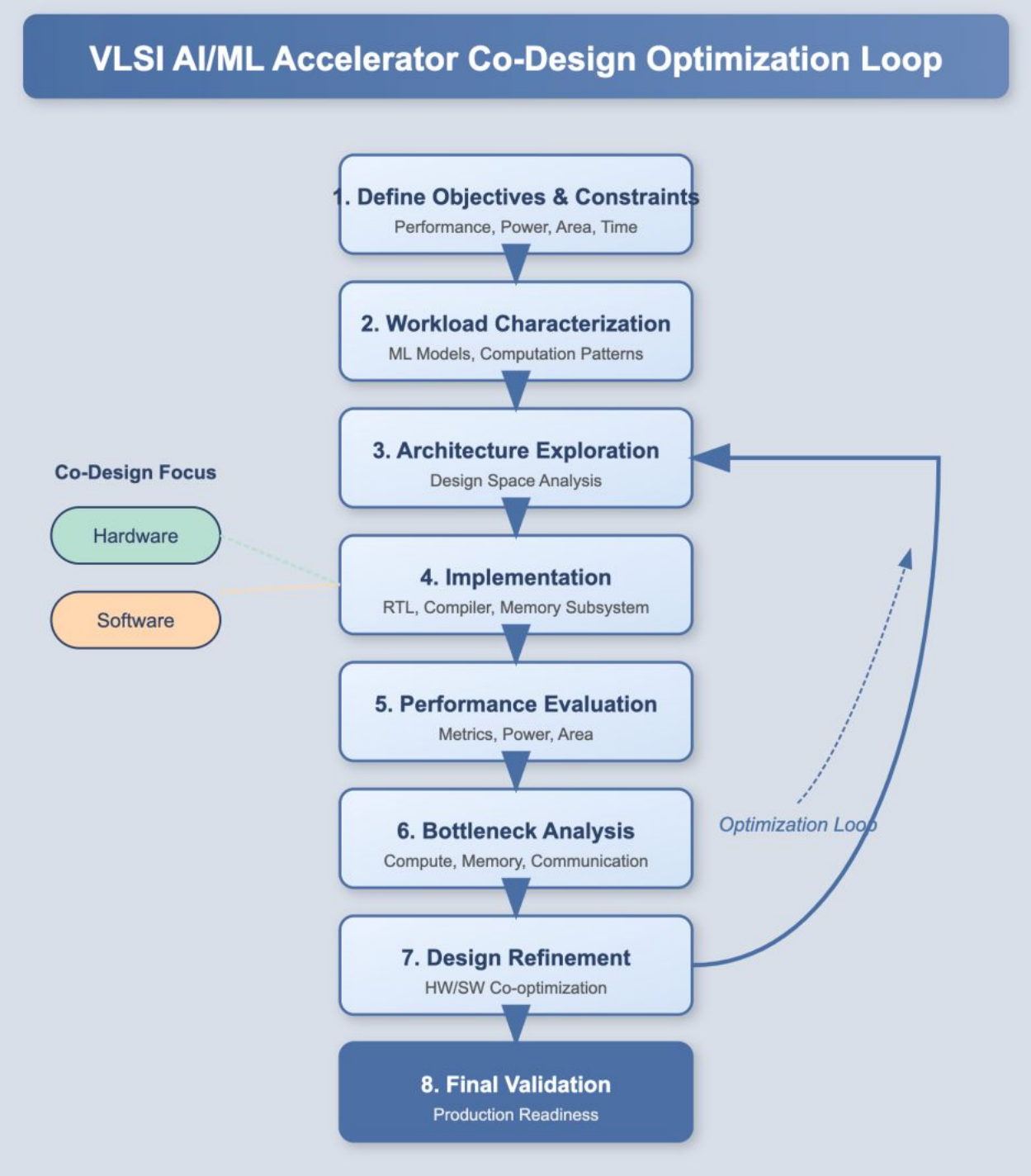
Integrated verification strategy

- Develop multi-level testbenches (unit, block, system)
- Create golden models for functional verification
- Implement power and performance verification methodologies
- Use hardware-in-the-loop testing with actual ML workloads

Hardware-aware algorithm optimization

- Adapt your algorithm/model to match hardware capabilities
- Implement quantization strategies optimized for your hardware
- Explore pruning techniques that leverage hardware efficiencies

Co-design optimization loop [More advanced]
Consider parametrizing your algorithm, HW design, and more, then implement an optimization loop that searches for optimal configurations.



Christof Teuscher

ECE 410/510: Hardware for AI and ML

Codefest #6: Practice HW/SW co-design with design iterations

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

teuscher Lab
teuscher-lab.com

Portland State
UNIVERSITY

Weekly challenges

Challenge #19: Implement a binary LIF neuron

Learning goals:

- Learn how a binary Leaky Integrate-and-Fire (LIF) neuron works.
- Implement such a neuron in Verilog or any other HDL.

Binary LIF neuron:

The binary LIF neuron can be formulated as:

- State representation: The neuron's state S(t) is either 0 (not spiking) or 1 (spiking)
- Simplfied update rule:
  - Accumulate input: $P(t) = \lambda P(t-1) + I(t)$
  - Where P(t) is a potential variable, λ is a leak factor (between 0 and 1)
  - I(t) is the binary input at time t
- Threshold function:
  - $S(t) = 1$ if $P(t) \geq \theta$ (threshold)
  - $S(t) = 0$ otherwise
- Reset mechanism:
  - If $S(t) = 1$, then P(t) is reset to a lower value

Tasks:

1. Write a Verilog implementation of a simple binary LIF neuron (using the formulation above) with a single input.
2. Write a testbench that demonstrates the following scenarios:
   - Constant input below threshold
   - Input that accumulates until reaching threshold
   - Leakage with no input
   - Strong input causing immediate spiking

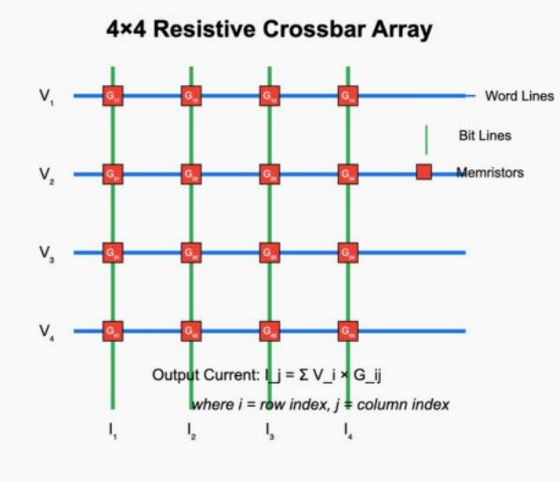
Challenge #20: Crossbar matrix-vector multiplication

Learning goals:

- Learn how to simulate a resistive crossbar in SPICE
- Learn how matrix-vector multiplication in a resistive crossbar works

Tasks:

1. Write SPICE code for a 4x4 resistive crossbar (with fixed resistances)
2. Demonstrate that the resulting output currents represent the product of the 4x1 input vector and the 4x4 weight matrix.



ASIC design flow

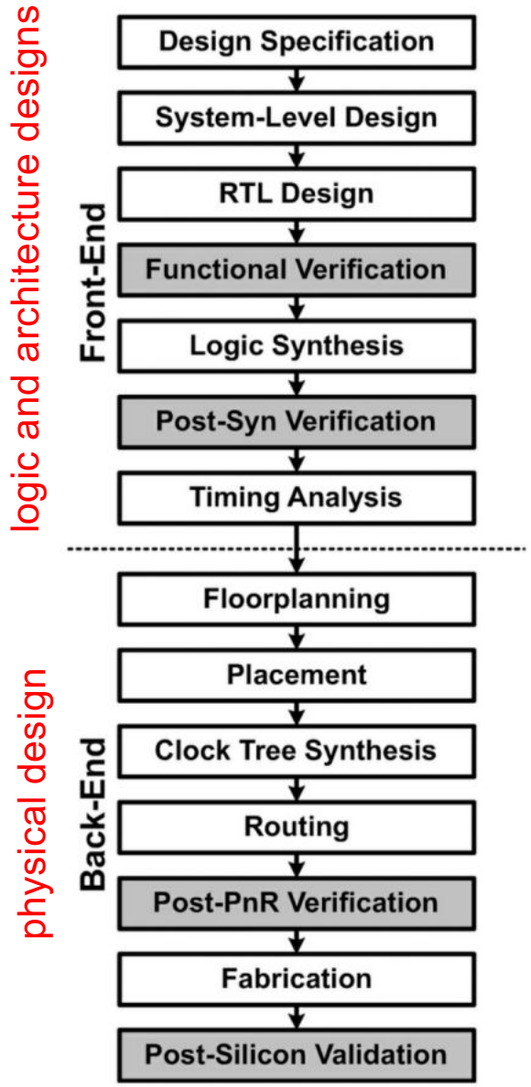


Fig. 9 ASIC design flow

Python
Verilog/SystemVerilog/VHDL

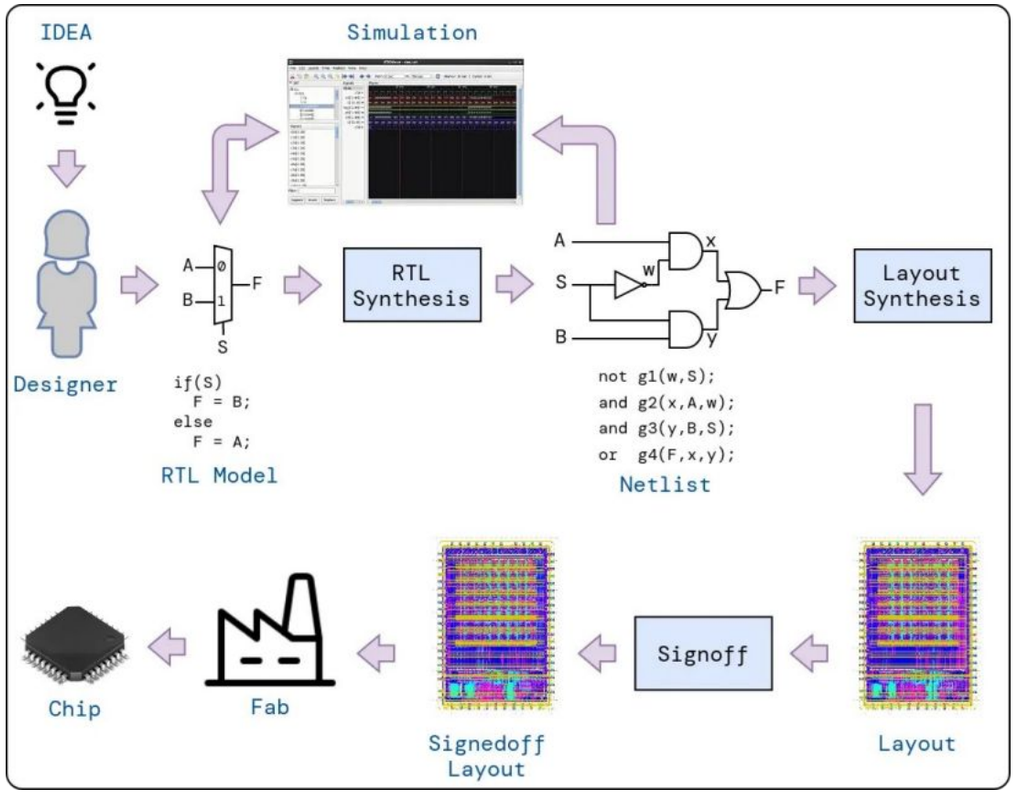
Max. frequency, # transistors, etc.

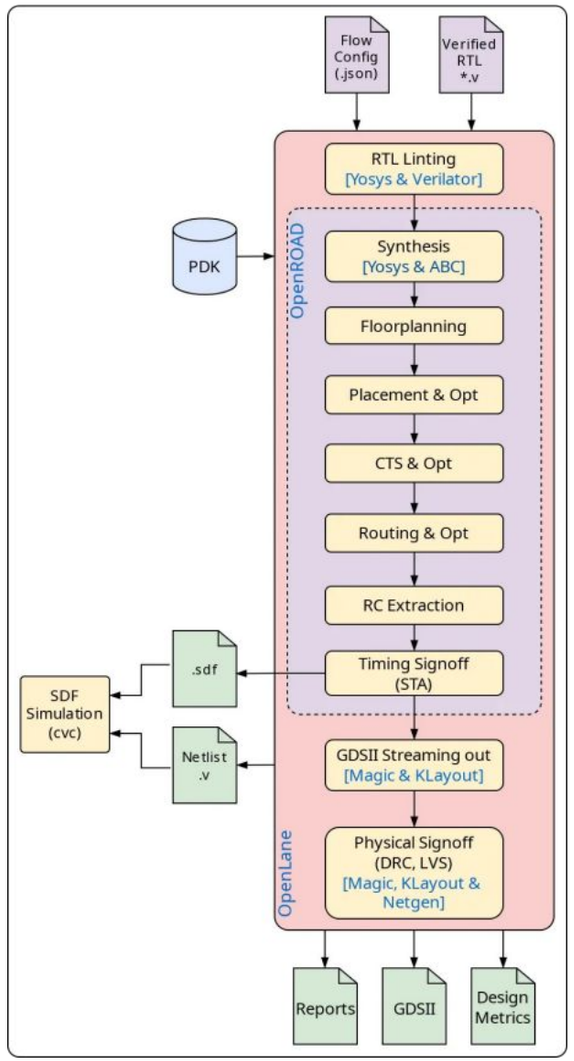
Mishra, Ashutosh; Cha, Jaekwang; Park, Hyunbin; Kim, Shiho. Artificial Intelligence and Hardware Accelerators.

teuscher Lab
teuscher-lab.com

Portland State UNIVERSITY

OpenLane 2 tools





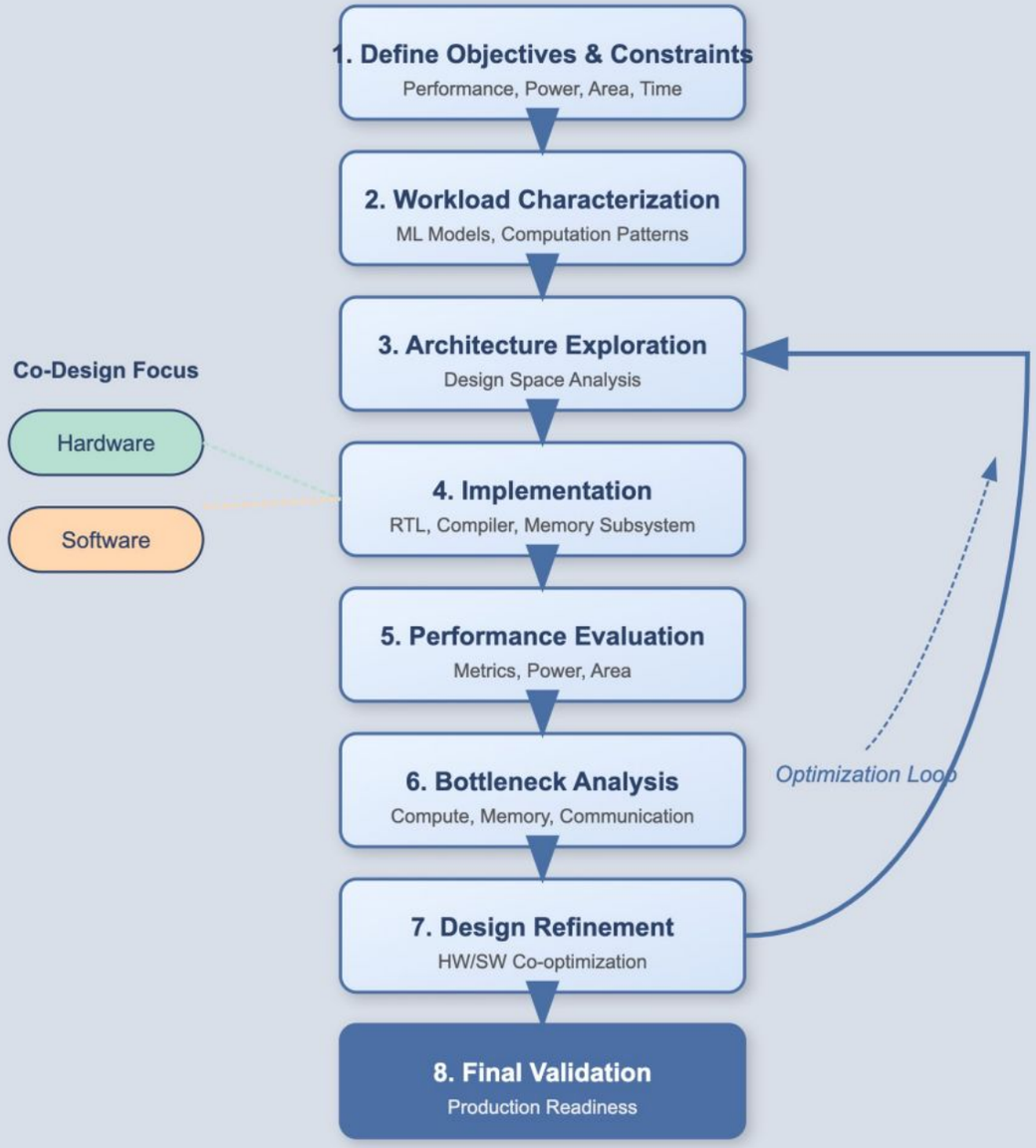
https://efabless.com/openlane
https://github.com/efabless/openlane2
https://openlane2.readthedocs.io/en/latest/getting_started/newcomers/index.html

Main goals for today

Based on your initial results, go back to the drawing board and do a design iteration.

https://www.pacific-research.com/iterative-product-development

VLSI AI/ML Accelerator Co-Design Optimization Loop



Today's presentations

Date	Time	Name
Wed, May 7	3:30: PM	Hippe, Brandon
Wed, May 7	3:32: PM	Kurra, Sai Sumanth
Wed, May 7	3:34: PM	Mastin, Tucker
Wed, May 7	3:36: PM	Knopp, Jack
Wed, May 7	3:38: PM	Desalegn, Melaku
Wed, May 7	3:40: PM	Malleboina, Nageswararao
Wed, May 7	3:42: PM	Jain, Alex
Wed, May 7	3:44: PM	Bandela, Vinod Kumar
Wed, May 7	3:46: PM	Routh, Vishwa Sree

Week 7

Challenges

ECE 410/510
Spring 2025

Instructions

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #22: Broadening your horizon about neuromorphic computing

Learning goals:

- Read a recent, state-of-the-art review paper on neuromorphic computing.
- Be able to digest the information and answer questions about the paper.

Tasks:

1. Read the following paper: Kudithipudi, D., Schuman, C., Vineyard, C.M. *et al.* **Neuromorphic computing at scale**. *Nature* **637**, 801–812 (2025). https://doi.org/10.1038/s41586-024-08253-8
2. Consider writing a "blog" post for your portfolio with your answers to the following questions:
    1. The authors discuss several key features necessary for neuromorphic systems at scale (distributed hierarchy, sparsity, neuronal scalability, etc.). Which of these features do you believe presents the most significant research challenge, and why? How might overcoming this challenge transform the field?
    2. The article compares neuromorphic computing's development to the evolution of deep learning, suggesting it awaits its own "AlexNet moment." What specific technological or algorithmic breakthrough might trigger such a moment for neuromorphic computing? What applications would become feasible with such a breakthrough?
    3. The authors highlight the gap between hardware implementation and software frameworks in neuromorphic computing compared to traditional deep learning. Develop a proposal for addressing this gap, specifically focusing on how to create interoperability between different neuromorphic platforms.
    4. The review emphasizes the importance of benchmarks for neuromorphic systems. What unique metrics would you propose for evaluating neuromorphic systems that go beyond traditional performance measures like accuracy or throughput? How would you standardize these across diverse neuromorphic architectures?
    5. How might the convergence of emerging memory technologies (like memristors or phase-change memory) with neuromorphic principles lead to new computational capabilities not possible with traditional von Neumann architectures? What specific research directions seem most promising?

Week 7

Codefest: Rethinking, reorienting, scaling back?

ECE 410/510
Spring 2025

Challenge #23

Overview and context:
Many of you started off with a project that was perhaps a little too ambitious. You may now realize that things won't exactly lead to success unless you change your approach. That's a great insight to have and part of science, engineering, and life, where things rarely are a straight line.

Learning goals:

- Remind yourself of the project requirements and how success is defined.
- Learn how to rethink, reorient, and scale back your project so you can declare success at the end.
- Apply co-design principles in doing so.
- Rethink and refine your LLM prompt engineering

Project requirements [REMINDER]:
See also https://docs.google.com/document/d/1_HSDXhJEF1F2Qu76zJOlygvPm_F9NqX5VZ0f8CGXI34

Goals:

1. Design, test, and benchmark a co-processor chiplet that accelerates parts of some AI/ML code/algorithm of your choice. Start with a blank slate for your design.
2. [ALTERNATIVE] Design a stand-alone chiplet. Your chiplet could, for example, include an ARM core.
3. You will be deciding what design constraints to impose (e.g., power budget) and what metrics to use (e.g., throughput).
4. Apply HW/SW co-design principles to decide what part of your algorithm is executed in HW.
5. The chiplet should be described in a HW-description language (e.g., Verilog). The design should be synthesizable in order to obtain your relevant metrics.
6. The deeper you can go into the HW design, the better. The ultimate goal is to go all the way down to an ASIC description. E.g., by using OpenLane 2's workflow to convert HDL into GDS. See codefest assignments for additional info, tools, hints, etc.

How is project success defined?

- 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.

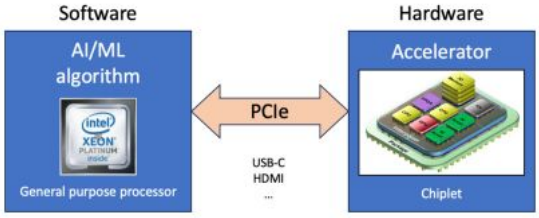


Figure 1. Applying a system-level co-design approach where the SW algorithm informs the HW.

Rethink your prompt engineering:

Core Principles

- **Be specific about the design stage** - Specify whether you need help with architecture exploration, RTL design, verification, physical design, or post-silicon validation
- **Provide technical context** - Include details about technology node, design constraints, and performance targets
- **Use domain-specific terminology** - Utilize standard chip design terminology to ensure clarity (HDL, DRC, timing constraints, etc.)

Effective Strategies

1. **Break down complex tasks** - Instead of asking for an entire chip design, focus on specific modules or functional blocks
2. **Specify design constraints clearly** - Include power, performance, area (PPA) requirements and other key parameters
3. **Request step-by-step approaches** - For complex design problems, ask for methodical solutions
4. **Leverage visual aids** - Request diagrams or suggest formats for outputs (block diagrams, timing diagrams, etc.)
5. **Create test cases and test benches** - For any design solution, systematically create test cases and test benches.

Example Prompts

- **Poor prompt:** "Design a chip for me."
- **Better prompt:** "Help me create an architectural block diagram for a low-power IoT SoC in 28nm technology with the following specifications: CPU core (ARM Cortex-M4), 256KB SRAM, Bluetooth LE radio, and temperature/humidity sensors. Target power consumption should be under 10mW in active mode."
- **Poor prompt:** "Check if my design works."
- **Better prompt:** "Review my SystemVerilog testbench approach for verifying a RISC-V processor cache controller. I need to validate coherence protocols across multiple cache levels. What assertions should I include to verify proper handling of modified cache lines during eviction? Please suggest specific SVA properties I could implement."
- **Poor prompt:** "How do I make my chip use less power?"
- **Better prompt:** "I'm designing a 7nm mobile SoC with dynamic voltage and frequency scaling. My current power analysis shows excessive leakage in the memory subsystem (approximately 25mW at idle). Suggest three specific techniques to reduce leakage power without compromising access time. Include both circuit-level and architectural approaches."
- **Poor prompt:** "Help with my chip layout."
- **Better prompt:** "I'm working on floorplanning a 5nm FinFET design with 12 processing cores and an on-chip network. My current approach places memory blocks at the periphery, but I'm seeing timing violations on critical paths. Considering IR drop and thermal constraints, what alternative floorplan organization would you recommend? Please explain the tradeoffs with respect to timing closure and signal integrity."
- **Poor prompt:** "Write Verilog code for a processor."
- **Better prompt:** "Help me implement a parameterized FIFO buffer in SystemVerilog with configurable depth and width. I need support for almost-full and almost-empty flags, with threshold values set at compile time. The design will be synthesized for an FPGA target (Xilinx Ultrascale+), so please optimize for resource utilization rather than timing. Include comments explaining synchronization mechanisms."
- **Poor prompt:** "Design an amplifier."
- **Better prompt:** "I need to design a low-noise amplifier for a 5G RF front-end operating at 28GHz in a 22nm FD-SOI process. Target noise figure is <2dB with IIP3 >10dBm while consuming less than 15mW. Suggest a circuit topology that meets these requirements, explaining key design considerations for biasing and matching networks. Include typical component values."

Week 8

Challenges

ECE 410/510
Spring 2025

Instructions

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #24: Run a simulation on the EBRAINS BrainScaleS-2 neuromorphic hardware

EBRAINS (https://www.ebrains.eu) is an open research infrastructure that gathers data, tools and computing facilities for brain-related research. The project is funded by the EU and the Human Brain Project

Learning goals:

- Run a simple simulation on the EBRAINS BrainScaleS-2 hardware.
- Explore the capabilities of BrainScaleS-2 as well as PyNN (https://neuralensemble.org/PyNN)

Tasks:

1. Request a free account for the EBRAINS neuromorphic platform at https://wiki.ebrains.eu/bin/view/Collabs/neuromorphic/Getting%20access
2. Once you have an account and a collab, head over to the BrainScaleS-2 demo code and descriptions at https://electronicvisions.github.io/documentation-brainscales2/latest/brainscales2-demos/tutorial.html
3. Pick one of the examples/demos, e.g., matrix multiplication, and run it on the BrainScaleS-2 hardware: https://electronicvisions.github.io/documentation-brainscales2/latest/brainscales2-demos/tp_00-introduction.html
4. Enjoy the excitement ☺!

Week 8

Codefest: Building and co-simulating an SPI interface in Python and Verilog

ECE 410/510
Spring 2025

Challenge #25

Overview and context:
Tomlinson, Li, and Andreou used an *Serial Peripheral Interface* (SPI) in their paper on "*Designing Silicon Brains using LLM: Leveraging ChatGPT for Automated Description of a Spiking Neuron Array*" to interface with the hardware: https://doi.org/10.1109/CAE59785.2024.10487167
SPI is a conceptually simple bus and one of the most widely used interfaces between microcontrollers and peripheral ICs. The throughput of an SPI bus can be 10Mbps and higher.

The above authors used the following prompts to design SPI modules:

- *I want you to now create a SPI interface to communicate with the network module above.*
- *Can you create a top file to connect this SPI module with the network module?*

Learning goals:

- (Vibe) code an SPI interface between your SW (e.g., Python) and HW (e.g., Verilog).
- Do a high-level co-simulation of the HW and SW and benchmark the system-level performance.
- Use cocotb for the co-simulation.

cocotb (reminders):

- https://www.cocotb.org
- cocotb is an open source coroutine-based cosimulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.
- cocotb works with any hardware design that your preferred simulator (and cocotb supports all major simulators) can simulate – be it in (System)Verilog, VHDL, a mixed language, or even a mixed-signal design.
- With cocotb, you write testbenches and verification code in Python. In addition to all the goodies of the Python programming language and its ecosystem, cocotb provides a lean framework to efficiently write verification code .

Tasks:

1. Either find an SPI IP module, design your own, or create one using vibe coding.
    a. My Claude prompt: "*Can you design an SPI interface that interfaces with a software component in Python and a Verilog component?"*
    b. Claude automatically suggested and generated a cocotb SPI interface, which allows to directly control Verilog signals.
    c. You may have to explicitly ask your LLM to do that.
2. Use cocotb to simulate and test the SPI.
    a. My Claude prompt: "*How would I test this in a combined simulation?"*
    b. Claude then generated an entire test suite, include setup scripts and makefiles.
3. Determine the throughput and latency of the SPI interface.
    a. My Claude prompt: "*Can you write a script that measures the throughput and the latency of the SPI interface?*"
    b. Claude generated (1) a standalone Python tool that can measure real-world performance metrics of your SPI interface in various configurations and (2) an advanced cycle-accurate performance analysis cocotb by hooking directly into Verilog.

```
SPI BENCHMARK SUMMARY
============================================================
Average Latency: 32.45 µs
Maximum Throughput: 3850.21 kbps
SPI Clock Frequency: 4.96 MHz
Protocol Overhead: 12.67 µs

Performance by Packet Size:
Size (bytes)    Throughput (kbps)    Efficiency (%)
------------------------------------------------------
1               320.12               77.52
8               1240.45              79.87
16              2005.32              82.43
32              2680.76              84.92
64              3150.45              86.73
128             3450.67              87.85
256             3677.33              88.54
512             3780.12              89.12
1024            3850.21              89.43
```

Figure 1: Benchmark summary example. Efficiency shows the percentage of time is spent on actual data transfer vs. overhead.

Week 9

Challenges

ECE 410/510
Spring 2025

Instructions

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #26: BrainChip's IP for Targeting AI Applications at the Edge

Learning goals:

- Learn more about BrainChip's approach to build IP cores and market their Akida chips for AI applications at the edge.
- Learn more about BrainChip's *Temporal Event-based Neural Network* (TENN) architecture and how that benefits AI at the edge.

Tasks:

1. Listen to the EETimes BrainChip podcast (47min) at https://www.eetimes.com/podcasts/brainchips-ip-for-targeting-ai-applications-at-the-edge
2. Put what you learned in perspective in a short write-up. In particular, compare the BrainChip approach to GPUs and other neuromorphic chips that we have seen in class.

Week 9

Codefest: Test, verify, and benchmark

ECE 410/510
Spring 2025

Challenge #27

Overview and context:
Verification and validation are often confused, but they answer fundamentally different questions about a design:

- Verification: "Are we building the product right?" Verification checks whether the design correctly implements the specified requirements. It's about correctness of implementation.
- Validation: "Are we building the right product?" Validation ensures the design meets the actual user needs and intended purpose. It's about fitness for purpose.

Modern testing philosophy: "Verification is not about proving there are no bugs—it's about gaining sufficient confidence that the design will work correctly in its intended application." The goal isn't perfection (which is impossible) but rather achieving an acceptable level of quality and reliability for the specific use case. Critical systems require more rigorous testing than consumer products, but all designs benefit from systematic verification. In essence, testing and verification transform designs from "it might work" to "we're confident it will work," which is essential for any serious engineering project.

Best practices for testing:

1. Constrained Random Testing: Use Python's random libraries with constraints
2. Coverage Tracking: Monitor which design features have been tested
3. Assertion Checking: Implement both Python and SystemVerilog assertions
4. Performance Metrics: Measure simulation speed and optimize bottlenecks
5. Regression Testing: Build a test suite that can be run automatically

Learning goals:

- Test, verify, and benchmark your chiplet design in a high-/low-level co-simulation (e.g., by using cocotb, see challenge #25)
- Compare it with your software implementation.

10-Step Summary: Testing Python-Verilog Designs with cocotb:
Example code available at https://drive.google.com/file/d/104PIB59HMKCW1M5zlg_SZw-xQWTdnVWe/view?usp=sharing

Step 1: Set Up the Cocotb Environment

- Install cocotb and required simulators (Icarus, Verilator, etc.)
- Create Makefile with simulation parameters
- Define testbench structure and clock generation
- Establish reset sequences and initial test connectivity
- Configure timing units and precision

Step 2: Create Functional Tests

- Define transaction classes for structured stimulus
- Test basic input/output functionality
- Verify core features work as specified
- Implement assertion-based checking
- Test each module feature systematically

Step 3: Implement Constrained Random Verification

- Create randomized transaction generators
- Define constraints for valid input ranges

- Build drivers to send transactions to *Design under Test* (DUT)
- Implement monitors to observe outputs
- Generate diverse test scenarios automatically

Step 4: Add Coverage Metrics
- Track which features have been tested
- Define coverage points for critical signals
- Monitor state machine transitions
- Create cross-coverage between related features
- Generate coverage reports and identify gaps

Step 5: Create Python Reference Model
- Build golden model of expected behavior
- Implement same algorithms in Python
- Compare DUT outputs against reference
- Detect functional mismatches automatically
- Maintain model synchronization with DUT

Step 6: Benchmark Performance
- Measure transaction throughput
- Calculate processing latency
- Track simulation speed metrics
- Identify performance bottlenecks
- Compare against design requirements

Step 7: Set Up Regression Testing
- Create parameterized test suites
- Automate test execution across configurations
- Track test results over time
- Generate regression reports
- Detect functionality degradation

Step 8: Add Debug and Visualization
- Capture signal traces during simulation
- Generate waveform visualizations
- Export VCD files for waveform viewers
- Create custom debug monitors
- Plot performance metrics graphically

Step 9: Test Corner Cases
- Test overflow/underflow conditions
- Verify reset behavior during operations
- Test simultaneous operations conflicts
- Check clock domain crossing safety
- Validate error handling mechanisms

Step 10: Final Validation and Report Generation
- Run complete test suite systematically
- Compile all test results
- Analyze coverage completeness
- Document performance measurements
- Generate comprehensive validation reports
- Provide pass/fail recommendations
- Archive results for future reference

Christof Teuscher

ECE 410/510: Hardware for AI and ML

Codefest #9: Test, verify, and benchmark

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

teuscher Lab
teuscher-lab.com

Portland State
UNIVERSITY

Weekly challenges

Challenge #26: BrainChip's IP for Targeting AI Applications at the Edge

Learning goals:

- Learn more about BrainChip's approach to build IP cores and market their Akida chips for AI applications at the edge.
- Learn more about BrainChip's *Temporal Event-based Neural Network* (TENN) architecture and how that benefits AI at the edge.

Tasks:

1. Listen to the EETimes BrainChip podcast (47min) at https://www.eetimes.com/podcasts/brainchips-ip-for-targeting-ai-applications-at-the-edge
2. Put what you learned in perspective in a short write-up. In particular, compare the BrainChip approach to GPUs and other neuromorphic chips that we have seen in class.

Week 9

Codefest: Test, verify, and benchmark

ECE 410/510

Spring 2025

Challenge #27

Overview and context:

Verification and validation are often confused, but they answer fundamentally different questions about a design:

- Verification: "Are we building the product right?" Verification checks whether the design correctly implements the specified requirements. It's about correctness of implementation.
- Validation: "Are we building the right product?" Validation ensures the design meets the actual user needs and intended purpose. It's about fitness for purpose.

Modern testing philosophy: "Verification is not about proving there are no bugs—it's about gaining sufficient confidence that the design will work correctly in its intended application." The goal isn't perfection (which is impossible) but rather achieving an acceptable level of quality and reliability for the specific use case. Critical systems require more rigorous testing than consumer products, but all designs benefit from systematic verification. In essence, testing and verification transform designs from "it might work" to "we're confident it will work," which is essential for any serious engineering project.

Best practices for testing:

1. Constrained Random Testing: Use Python's random libraries with constraints
2. Coverage Tracking: Monitor which design features have been tested
3. Assertion Checking: Implement both Python and SystemVerilog assertions
4. Performance Metrics: Measure simulation speed and optimize bottlenecks
5. Regression Testing: Build a test suite that can be run automatically

Learning goals:

- Test, verify, and benchmark your chiplet design in a high-/low-level co-simulation (e.g., by using cocotb, see challenge #25)
- Compare it with your software implementation.

Today's presentations

1 slide, 1 minute. Strict!

Wed, May 28	3:30: PM	Pallappa, Jaswanth
Wed, May 28	3:32: PM	Ramesh, Lokarjun
Wed, May 28	3:34: PM	Sarangdhar, Nitin
Wed, May 28	3:36: PM	Weeks, Stephen
Wed, May 28	3:38: PM	Bala Baskaran, Thanuja
Wed, May 28	3:40: PM	Chunduri, Bhargav
Wed, May 28	3:42: PM	Gopavarapu, Ramakrishna
Wed, May 28	3:44: PM	Cox, Riley
Wed, May 28	3:46: PM	Seggari, Koushik

Week 10

Challenges

ECE 410/510
Spring 2025

Instructions

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- I'd suggest you try to solve at least one problem per week. More is obviously better.
- Practice "vibe coding" if necessary.
- Post your solution(s) in the `#weekly-challenges` Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make your code available on Github.

Challenge #28: Model and simulate a memristor

Memristors are very important circuit elements of emerging neuromorphic hardware because they can directly emulate a synapse. Their resistive value represents the synaptic weight, while the weight can be changed by, for example, using a *Spike Timing-Dependent Plasticity* (STDP) rule (see lecture slide for details).

Learning goals:

- Learn how to model and simulate a memristor, either in Python or in SPICE

Tasks:

1. Download or write your own memristor model in SPICE or Python. E.g., you can use the Biolek model: https://www.radioeng.cz/fulltexts/2009/09_02_210_214.pdf
2. Visualize the characteristic pinched hysteresis loop in the I-V curve.
3. Document your results.

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.

Christof Teuscher

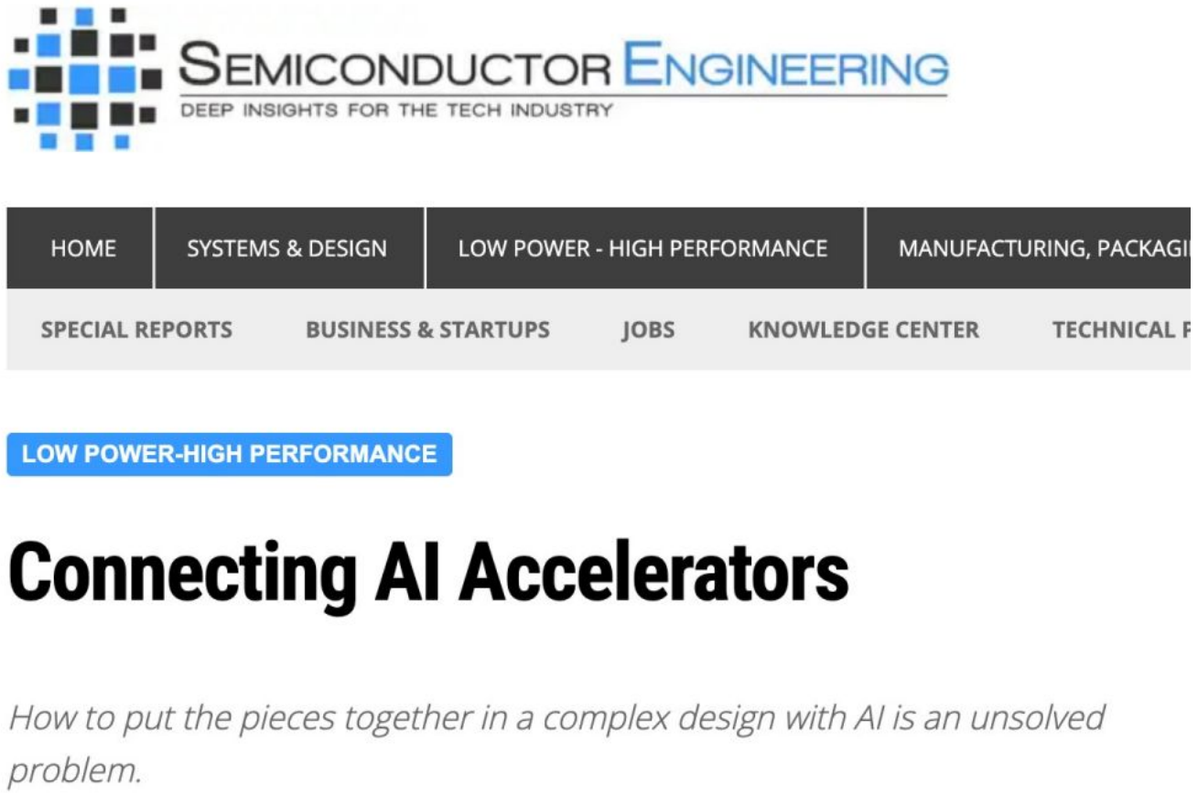
ECE 410/510: Hardware for AI and ML

Codefest #10: Hello doc(umentation)!

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

teuscher Lab
teuscher-lab.com

Portland State
UNIVERSITY



UCIe (Universal Chiplet Interconnect Express) is an open industry standard for connecting chiplets within a multi-die package

https://semiengineering.com/connecting-ai-accelerators

Weekly challenges

Challenge #28: Model and simulate a memristor

Memristors are very important circuit elements of emerging neuromorphic hardware because they can directly emulate a synapse. Their resistive value represents the synaptic weight, while the weight can be changed by, for example, using a *Spike Timing-Dependent Plasticity* (STDP) rule (see lecture slide for details).

Learning goals:

- Learn how to model and simulate a memristor, either in Python or in SPICE

Tasks:

1. Download or write your own memristor model in SPICE or Python. E.g., you can use the Biolek model: https://www.radioeng.cz/fulltexts/2009/09_02_210_214.pdf
2. Visualize the characteristic pinched hysteresis loop in the I-V curve.
3. Document your results.

Project success reminders

Goals

- Design, test, and benchmark a co-processor chiplet that accelerates parts of some AI/ML code/algorithm of your choice (see list at the end). Start with a blank slate for your design.
- [ALTERNATIVE] Design a stand-alone chiplet. Your chiplet could, for example, include an ARM core.
- You will be deciding what design constraints to impose (e.g., power budget) and what metrics to use (e.g., throughput).
- Apply HW/SW co-design principles to decide what part of your algorithm is executed in HW.
- The chiplet should be described in a HW-description language (e.g., Verilog). The design should be synthesizable in order to obtain your relevant metrics.
- The deeper you can go into the HW design, the better. The ultimate goal is to go all the way down to an ASIC description. E.g., by using OpenLane 2's workflow to convert HDL into GDS. See codefest assignments below for additional info, tools, hints, etc.

How is project success defined?

- 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.

If you did all the above, and have all your code on Github, created a solid portfolio that not only documents your final design, but also how you got there, how you gradually improved your design, and there is no evidence that you simply copied or replicated a solution from somewhere, then you should be in a good position to claim an A for yourself in this course.

Dos (only a selection)

- Project overview, architecture diagram, quick-start guide.
- Distinct SW/HW documentation. All files should have headers.
- All design decisions must be justified thoroughly. Explain alternatives and trade-offs you considered. I want to understand how you ended at your final solution.
- Include AI prompts and acknowledge the use of AI.
- Document testing and verification scenarios.
- 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.

Today's presentations

1 slide, 1 minute. Strict!

Wed, Jun 4	3:30: PM	Manikyanahalli Srinivasegowda, Bhavana
Wed, Jun 4	3:32: PM	Kanipogu, Balu Anudeep
Wed, Jun 4	3:34: PM	Knapp, Jon
Wed, Jun 4	3:36: PM	Deokar, Satyajit
Wed, Jun 4	3:38: PM	Alshaiji, Mohammad
Wed, Jun 4	3:40: PM	Cardenas, Hector
Wed, Jun 4	3:42: PM	Patil, Siddesh Dinesh
Wed, Jun 4	3:44: PM	Fraly, Nathan

Week 1

Challenges

ECE 410/510
Spring 2025

Instructions:

- The challenges below are for you to delve deeper into the subject matter and to test your own knowledge.
- Try to solve at least one problem per week. More is obviously better.
- Post your solution(s) in the #weekly-challenges Slack channel so everybody can appreciate what you did, ask questions, and make comments.
- Document everything for your portfolio and make any code available on Github.

Challenge #1

Go to the IEEE International Roadmap for Devices and Systems (IRDS) website (https://irds.ieee.org) and browse through the "Beyond CMOS" 2023 update roadmap to get a sense of the variety of devices that are being considered for beyond Moore technologies:
https://irds.ieee.org/images/files/pdf/2023/2023IRDS_BC.pdf
Read as long as your interest lasts.

Challenge #2

Read the following paper:
James P. Crutchfield, William L. Ditto, Sudeshna Sinha; Introduction to Focus Issue: Intrinsic and Designed Computation: **Information Processing in Dynamical Systems—Beyond the Digital Hegemony**. *Chaos* 1 September 2010; 20 (3): 037101. https://doi.org/10.1063/1.3492712

Challenge #3

Use the power of the internet and of LLMs to identify a physical system that solves differential equations inherently, through its physical properties, without executing instructions as a traditional processor does.

Week 1

Codefest: Warm-up

ECE 410/510

Spring 2025

Before you start

1. If you don't have a Github yet, create one where you'll be hosting all your code.
2. Make the repository public so that your peers can access your code.
3. Decide what platform you want to use do document everything you produce in this class. The platform of your choice must be public and accessible to your peers.
    a. Blog (e.g., Wix, Wordpress, Squarespace, GoogleSites)
    b. Technical reports on Github
    c. Github wiki
    d. …
4. **Submit your Github and platform choice(s) on Canvas by Sun, Apr 6, 11:59pm.**

During the codefest:

- Slack will be our main collaborative platform (at least for now).
- Post questions, solutions, insights in the #codefests channel.
- Present what you've got, whether it works or not.

Challenge #4

Learning goals:

- Experiment with LLM-assisted chip design
- Do "vibe coding" and experience the problems associated with it.
- Install and test the entire workflow/toolchain.

Tasks:

1. Replicate the Johns Hopkins paper:
    a. **Designing Silicon Brains using LLM: Leveraging ChatGPT for Automated Description of a Spiking Neuron Array**
    b. https://arxiv.org/abs/2402.10920
2. Use your favorite LLM.
3. Experiment with the queries and see what happens.
4. Keep track of all the queries you made that got you to the finished result.
5. If you want to go all the way down to the ASIC, you can use OpenLane (https://www.zerotoasiccourse.com/terminology/openlane) to convert HDL into GDS (used for ASICs).
6. Compare the results of your version with their paper.
7. Can you think of any improvements to their solution?
8. Can you do a design with a RLU instead if a LIF neuron? https://en.wikipedia.org/wiki/Rectifier_(neural_networks). Or a Hodgkin–Huxley neuron model?
9. Document your results and findings carefully. What did you learn?

Challenge #5

Learning goals:

- Learn how to analyze and profile AI/ML, and other workloads, e.g., written in Python
- Learn how to identify bottlenecks and parallelism
- Learn how to think about candidate execution architectures.
- Do "vibe coding" and experience the problems associated with it.

Tasks:

1. Pick 3 different Python programs/workloads. E.g.,
    a. Differential equation solver
    b. Convolutional neural network
    c. Traveling Salesman Problem (TSP)
    d. Quicksort
    e. Matrix multiplication
    f. A cryptography algorithm, e.g., AES
    g. …
2. Either write your own code (probably not enough time), download some code, or ask your LLM to generate examples.
3. Compile the code into Python bytecode. Ask your LLM how to do that. Or look it up. Hint: `py_compile`.
4. Disassemble the bytecode and look at the instructions. Hint: `dis`
1. Can you guess what virtual machine Python uses just by looking at the bytecode?
2. How many arithmetic instructions are there? Hint: http://vega.lpl.arizona.edu/python/lib/bytecodes.html
3. Write a script that counts the number of each instruction. Hint: ask your LLM.
4. Compare the instruction distribution for your 3 workloads.
5. Use a profiler to measure the execution time and resource usage of your codes. Hint: `cProfile`. Hint: `snakeviz` allows for interactive visualization.
6. Ask your LLM to write you some code to analyze the algorithmic structure and data dependencies of your code to identify potential parallelism.
7. Now, knowing all these details, what instruction architectures would you build for each of these workloads?
10. Document all your findings and insights carefully. What did you learn?