Electronic Design

By GILLES BACKHUS, Co-Founder and VP of AI,
Recogni, www.recogni.com

Hardware/Software Co-Design: The Five Core Principles

Developing a solution across geographic time zones, cultures, and skillsets can be difficult. Following these core principles to optimize hardware and software system components can help maximize success from the outset.

Hardware/software co-design attempts to optimize the hardware and software components of a complex electronic system while satisfying the project's goals and design constraints, which usually encompass performance, cost, and power consumption.

This article discusses the five core principles that helped Recogni build a geographically diverse engineering organization that successfully developed a machine-learning (ML) chip while meeting ambitious performance, power-consumption, and development schedule goals. ML software is a rather extreme software case because of the nondeterministic nature of ML development. Experiences with this unique software niche underscore the critical nature of several of these five core principles.

The development team has always had two homes: San Jose, California and Munich, Germany. The San Jose team owns silicon and hardware design, firmware and software development, and data capture and creation. The Munich team develops ML-related software, including the compiler and the necessary software infrastructure ranging from training datasets to compiled perception stacks.

This geographic separation amplifies the need for, and the difficulties in making, the worlds of hardware and software development mesh smoothly. It also makes the five core principles of hardware/software co-design stand out in even bolder relief.

Before discussing the five core principles of hardware/software co-design, it's important to discuss and understand the two different engineering personas—hardware and software—that underlie the team dynamics. In this case, the personas of our development team are somewhat representative of hardware/software co-design teams everywhere.

Our typical silicon designer is an experienced engineer who has successfully designed, tested, and brought up many chips in their career. These engineers are acutely aware of the "one shot" nature of their work. The consequences of major hardware design errors are very costly and include ballooning silicon design costs and loss of precious time to additional design iterations. Such consequences, which can kill a startup company, cause the hardware design team to lean toward a conservative design approach.

Conversely, our typical ML designer has five years of experience or less and is accustomed to working in a very organic, explorative, and forgiving development environment. Somehow, these two worlds—hardware and software—must work together harmoniously to produce the desired product on time and within budget.

In some fundamental ways, these two personas are quite different, and it can be very difficult to mesh them into one focused team. However, forging these two diverse engineering personas into one coherent engineering team that works in synchronicity proved to be key to developing a new and innovative piece of ML silicon in minimal time. The five core principles guided the building and management of our team:

Co-Design Core Principle #1: Determinism

The first core principle of hardware/software co-development is determinism. One way to look at determinism is to ask this question: How confident are you about the predictability of a new task's achievability (such as implementing a new feature or a new circuit)?

Some engineering disciplines—including full-stack software engineering, standard CMOS chip design, and PCB design—operate with high levels of determinism. High levels of determinism mean that when there's a new engi-

neering challenge to tackle, an experienced engineer or engineering manager can confidently predict a development timeline, saying "I have done this before" or, at least, "A guy on reddit has implemented something similar."

High levels of determinism don't imply that the task is simple to achieve. The task may be very challenging and might require a very skilled engineer to execute it. However, high levels of determinism mean that the engineer or engineering manager has a much clearer idea of how long a task will require for completion. In other words, the task's dependencies and critical paths are well understood.

Conversely, some engineering disciplines (including ML engineering) work day-to-day with a general feeling of nondeterminism. Even though a neural-network stack is technically deterministic—the same input always results in the same output—the determinism of the final product is submerged in the sheer, ungraspable complexity and arbitrary behavior of ML networks. As a result, ML network development requires a very iterative approach, not by choice but out of necessity.

Friction can result when parts of a hardware/software co-design team work on tasks with high determinism and other parts of the team work on tasks with low determinism. Overcoming this potential source of friction boils down to empathic interaction. Both sides of the development team need to have high degrees of empathy for the nature of the other side's work. Only then is highly synergistic and cohesive hardware/software co-design possible, which directly leads to core principle #2.

Co-Design Core Principle #2: Communication

Certainly, this isn't the first article to state the obvious importance of good communication among development team members. However, when discussing hardware/software co-design, we learned that a few specific aspects of team communications should be emphasized.

The first such aspect is the need for over-communication. When dealing with a group of people who think very differently, repetition is extremely important. Repeat, repeat, and repeat.

Just because someone mentioned the meaning of an acronym three weeks ago in a standup meeting doesn't guarantee that team members in other disciplines will have internalized the theoretical meaning and practical implications of that acronym. Even a simple word like "segmentation" can mean completely different things to a hardware chip designer versus an ML software engineer.

Context may eliminate confusion in most situations, but it's essential that members of closely intertwined, cross-disciplinary engineering teams take a step back and actively try to put themselves in the position of their counterparts on the team.

Yet another communications challenge is psychological in nature. To some team members, repetition may appear unnatural and almost condescending. Team members must fight the urge to avoid what might seem like an implicit confrontation or insult. The end goal should always be clarity and team members should be aware of, and make accommodations for, this when evaluating communications from other team members.

Another critical communications element is treating internal documentation as a product. The need for complete and clear documentation starts with the on-boarding of new team members. Requiring clear internal documentation sets the right tone at the very beginning and leads to a very cooperative and highly intertwined way of working.

Rock-solid documentation from both sides of the hardware/software co-development team must be easily accessible and easy to find. The guiding communications principle here is: Treat all your colleagues like close customers. Serve them; empathize with them; collaborate with them.

Co-Design Core Principle #3: Management

Many product development cycles contain critical paths consisting of tasks from different disciplines that might not synchronize well. This temporal mismatch poses a challenge. For example, in Recogni's experience, precisely time-lined productization doesn't pair well with ML's unavoidable and unpredictable need for iterative exploration and innovation.

It's not uncommon for an ML project's complexity to suddenly balloon from an easily-done-in-2-weeks, incremental, model-retraining task to a new, from-scratch project involving numerous aspects that must be researched over a period of months.

Solving this temporal mismatch requires two key ingredients:

- Project management must profoundly appreciate the respective natures and challenges of all involved disciplines and a slightly more conservative project timeline that accounts for significant critical-path disturbances.
- Technical leads managing mixed teams that include ML engineers need to understand that the devil lies in the details and that baby steps and reflecting on the challenges which come with abstraction layers are crucial to successful project completion.

Imagine, for instance, the silicon team wants to evaluate the value of writing support for a new instruction to support a new type of neural-network architecture. What should the ML team do? Test the new architecture, of course. However, in the ML world, things aren't as simple or obvious as they seem, initially.

For example, here's a sequence of events that happened to the Recogni development team. Discovery of a remotely related paper with code suggested that a specific ML stack

could benefit the target product. Even better, the target technology already under development could execute this new ML stack efficiently and effectively. At this point, the new ML stack from the paper looked like a natural fit.

However, one hyperparameter out of maybe more than 100 can completely ruin an ML model's performance and send the design team down a wrong path based on faulty conclusions. A non-obvious change of one hyperparameter value from 0.01 to 0.002 suddenly made the ML stack work, seemingly by magic. Welcome to the unpredictable ML world.

The solution to this challenge is to take baby steps, which many people highly underestimate, especially those new to ML or those who have spent their last several years in more deterministic and humanly graspable engineering disciplines. ML development teams that start big right away can drown in debugging. Most of the time, the quickest path to ML success is to start small.

Co-Design Core Principle #4: Beware of Abstraction

Any serious product-level ML development environment requires abstraction because it enables scalability. However, abstraction comes with the often-underestimated risk of wrongly implied automatism and genericness.

For example, the reason why the incorrectly set hyperparameter in Principle #3 might have been set to 0.01 could simply be because a module that's been part of the environment for a long time led developers to assume that the hyperparameter "just works" with that setting. After all, that setting literally did work for months or years, so it's not a bad assumption.

However, the long-term and successful use of this setting allowed the awareness of the ML model's significant sensitivity to that hyperparameter to fade away over time. It's economically unfeasible to test every ML hyperparameter setting every time you want to develop something new.

Constant and persistent reflection on the adaptability level for all abstraction layers, preferably supported by a range of randomized, recurrent, and automated end-to-end unit tests, helps to minimize the risks associated with abstraction.

Co-Design Core Principle #5: Scope and Focus

There's a good reason why we consciously and intentionally decided to create a company that builds end-to-end reference systems that span all aspects of a design from sensors in the car, to self-captured data, chip design, neural-network design, compiler tool chain, visualization, etc. This comprehensive scope enables our core teams to develop their respective parts of the stack in the context of a complete, end-to-end system instead of in isolation.

Due to the broad scope of the design, the team can observe the system-level implications of its work. However, the broadened scope delivers additional benefit: It creates a shared narrative for our teams internally. All hardware/software development teams need this shared narrative.

For instance, integrating an image sensor as part of a reference platform should, in theory, not be a difficult requirement for a company that's developing an ML processing chip.

For the hardware chip designers, the image sensor is a piece of embedded electronics that they must physically and logically interface with. For the ML perception engineer, the sensor serves as the data source for the neural network's input layer instead of using synthetic data sets that can lead to suboptimal designs. The physical sensor produces a real-world dataset that serves as a basis for empathic interaction between the hardware engineers and the ML developers.

Conclusion

These five core principles should be integrated into every hardware/software development project. As we've discovered at Recogni, they're especially important in the world of ML development. These core principles are designed to emphasize communications and empathy throughout the diverse development team, at the beginning of the project, throughout the project, and continuing all the way to the project's end.

Adhering to these principles doesn't guarantee success. There's lots of hard work between here and there. However, adopting these principles and embracing them gives hardware/software co-development teams a fighting chance to reach the end goal in a timely manner.

Gilles Backhus has spent his career focusing on real-time AI development, seeking to solve our world's most important technical challenges.